SAME OBJECTIVE, DIFFERENT MOMENTS: RESIDUAL-GUIDED TEMPORAL RESAMPLING FOR VLA CONTINUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Continued adaptation of task-adapted vision-language-action (VLA) policies seeks to improve performance under a finite update budget without inducing overfitting. Temporal resampling can reallocate this budget across trajectory moments, but without correction it also reweights the baseline continuation objective, making it difficult to determine whether performance gains arise from temporal allocation or objective reweighting. We introduce Residual-Guided Temporal Resampling (ReTime), which decouples temporal exposure from objective weighting. ReTime uses Flow-Matching residuals from a frozen starting policy to construct a non-uniform temporal proposal and applies importance correction to preserve the baseline continuation objective and its expected gradient. It modifies only the temporal sampling distribution, while keeping source and trajectory selection probabilities fixed. Across Florence and Qwen backbones in simulation and real-world settings, ReTime consistently achieves the strongest average performance. On Florence–SimplerEnv, 500 additional updates improve performance by 22.92 percentage points over the starting policy and by 11.20 points over the strongest continuation baseline. Over longer-horizon real-world continuation, ReTime continues to improve after Uniform begins to degrade. Mechanistic analysis further shows that, under matched temporal allocation, importance correction provides an additional 4.69-point gain on Florence–SimplerEnv, demonstrating that temporal exposure and objective weighting are not equivalent and motivating their explicit decoupling.

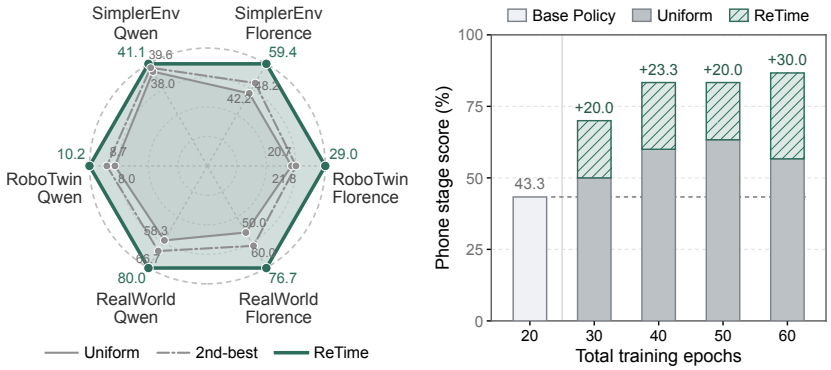


Figure 1: ReTime delivers the strongest finite-budget continued-adaptation performance across six simulation and real-world benchmark–backbone settings (Left), and this advantage extends to longer continuation-training horizons: on Florence–Phone, ReTime continues to improve even after Uniform begins to degrade with further continuation training (Right).

1 INTRODUCTION

Recent vision-language-action (VLA) models leverage large-scale vision-language pretraining and diverse robot demonstrations to learn transferable representations and generalizable control poli-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

cies (Brohan et al., 2023; Kim et al., 2025b; Black et al., 2025; Physical Intelligence, 2026). Task-specific post-training further adapts these generalist policies to downstream tasks or robot embodiments using domain-specific demonstrations (Black et al., 2024; Kim et al., 2025a). However, a single adaptation stage may not fully exploit the potential of the resulting policy, and further training on the same task data can still yield additional gains (Liu et al., 2026b; Jiang et al., 2026; Hancock et al., 2026; Physical Intelligence et al., 2025). This motivates *continued adaptation*: further training an already task-adapted policy under a fixed update budget. Because different moments within a demonstration may contribute unequally to optimization, the key question is how to allocate this limited budget across temporal positions.

A natural approach is to reallocate a fixed training budget across trajectory moments via temporal resampling. Existing methods do so through keyframe weighting (Wen et al., 2021), demonstration compression (Guo et al., 2025), transition filtering (Zhang et al., 2026), frame selection or remapping (Yu et al., 2026a), and loss-based importance sampling (Katharopoulos & Fleuret, 2018). However, non-uniform sampling simultaneously changes both the encounter frequency of each moment and its effective weight in the baseline continuation objective. Without correction, moments sampled more often are also upweighted in expectation, coupling *temporal allocation* with *objective reweighting*. As a result, it is unclear whether performance gains arise from improved temporal allocation, a shifted objective, or their interaction. This raises a central question: can we change *when* the policy trains while keeping *what* it optimizes unchanged?

We introduce *Residual-Guided Temporal Resampling* (ReTime), which decouples how often trajectory moments are sampled from how much they contribute to the baseline continuation objective. A frozen starting policy scores trajectory moments using Flow-Matching residuals, and moments with larger residuals are sampled more frequently. Importance correction compensates for this sampling shift, preserving the baseline continuation objective and its expected raw gradient. Although the baseline objective remains unchanged, the distribution of stochastic gradients can differ, allowing temporal allocation to influence optimization under a finite update budget.

Across six simulation and real-world backbone–benchmark settings spanning Florence and Qwen, ReTime achieves the highest average performance among the evaluated methods (Figure 1, left). On Florence–SimplerEnv, 500 additional updates improve performance by 22.92 percentage points over the starting policy and by 11.20 points over the strongest continuation baseline. Under matched temporal allocation, importance correction yields a further 4.69-point gain, showing that temporal allocation and objective treatment provide separable benefits. The advantage also persists beyond this short continuation budget: on Florence–Phone, Uniform eventually degrades, whereas ReTime continues to improve to 86.7% at 60 epochs, widening the gap to 30.0 points (Figure 1, right).

Our contributions are:

- We formulate continued VLA adaptation as temporal allocation under a fixed baseline continuation objective, separating the effects of temporal allocation and objective reweighting.
- We introduce ReTime, which uses residuals from a frozen starting policy for temporal resampling and importance correction to preserve the baseline continuation objective and its expected raw gradient.
- We demonstrate consistent improvements across six simulation and real-world settings and extended training horizons, with controlled analyses showing additional gains from objective preservation under matched temporal allocation.

2 Related Work

Allocation granularity in robot learning. Robot-learning pipelines allocate data at several granularities. Data curation filters unreliable state–action pairs or selects trajectories for behavioral coverage (Zhang et al., 2026; Wu et al., 2026), while continual-learning methods allocate replay across skills or semantic phases (Chen et al., 2026). Temporal methods instead emphasize informative moments through keyframes, frame remapping, or variable downsampling (Wen et al., 2021; Yu et al., 2026a; Guo et al., 2025). Related work also shows that apparent temporal benefits can arise from adding recovery-state coverage rather than reweighting existing observations alone (Liu et al., 2026a). We isolate the latter setting: the corpus, source and trajectory probabilities, observations, and action chunks are fixed, and only the conditional distribution over valid anchors within each

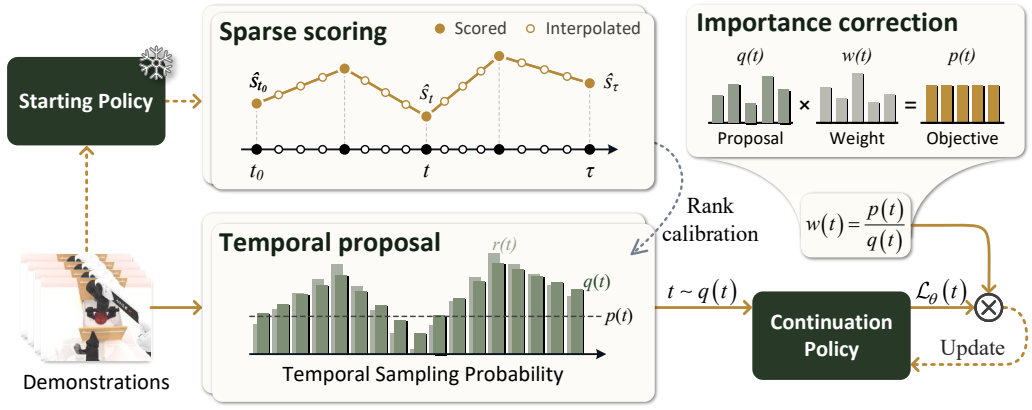


Figure 2: **ReTime framework.** A frozen copy of the starting policy estimates Flow-Matching residuals at sparse physical-time anchors. Linear interpolation, rank calibration, and a baseline mixture convert these scores into a full-support temporal proposal $q(t)$. The continuation policy samples anchors from $q(t)$ and weights each loss by $w(t) = p(t)/q(t)$, preserving the baseline continuation objective and its expected raw gradient.

trajectory changes. The resulting question is therefore not which data to collect, retain, or compress, but how often existing temporal supervision should be encountered under a finite continuation budget.

Model-conditioned sampling under a fixed objective. Curriculum learning, self-paced learning, active bias, and hard-example mining use difficulty or prediction error to allocate training effort (Bengio et al., 2009; Kumar et al., 2010; Chang et al., 2017; Shrivastava et al., 2016). Prioritized replay further combines error-based sampling with importance weights (Schaul et al., 2016), and diffusion models have used importance sampling across noise levels (Nichol & Dhariwal, 2021). In robot policy learning, VADF adapts sampling over diffusion timesteps and trajectories using online loss signals (Yu et al., 2026c). These methods establish the value of model-conditioned allocation, but they target different sampling units or do not isolate temporal exposure under the baseline continuation objective.

Importance sampling separates encounter frequency from objective weight through probability-ratio correction (Needell et al., 2014; Katharopoulos & Fleuret, 2018). ReTime applies this principle specifically to physical trajectory time within robot demonstrations: a frozen starting policy performs offline residual scoring once, a full-support temporal proposal reallocates temporal exposure, and ordinary p/q importance correction preserves the baseline continuation objective and its expected raw gradient while leaving Flow-Matching time and noise unchanged. This combination distinguishes our setting along three axes: within-trajectory allocation, model-conditioned residual scores, and objective-preserving continuation training. Uniform sampling and randomized score assignments then separate the effect of non-uniform exposure from alignment with the residual signal.

3 Method

3.1 Problem Formulation

With unweighted training losses, sampling a moment more often also increases its contribution to the baseline continuation objective. Our goal is to change sampling frequency while retaining the baseline objective weights.

Let d denote a data source, e a demonstration trajectory, and $t \in \mathcal{T}_e$ a valid physical-time anchor for an observation–action-chunk pair. The baseline sampling distribution is $p(d, e, t) = p(d)p(e \mid d)p(t \mid e, d)$, where the baseline temporal distribution is $p(t \mid e, d) = 1/N_e$ and $N_e = |\mathcal{T}_e|$. For per-example loss ℓ_θ and Flow-Matching variables $\xi \sim P_\xi$, the baseline continuation objective is

$$\mathcal{L}_p(\theta) = \mathbb{E}_{(d,e,t)\sim p,\ \xi\sim P_\xi}\left[\ell_\theta(d, e, t; \xi)\right]. \tag{1}$$

We modify only the temporal conditional, replacing $p(t \mid e, d)$ with a full-support temporal proposal $q(t \mid e, d) > 0$ on every valid anchor. The source and trajectory distributions, valid-anchor sets, and P_ξ remain unchanged.

3.2 ReTime: Residual-Guided Resampling

Figure 2 summarizes the ReTime pipeline. Residuals from a frozen starting policy are converted into a temporal proposal that reallocates within-trajectory training exposure, while importance correction compensates for the resulting sampling shift and preserves the baseline continuation objective.

Sparse residual scoring. We use a frozen copy of the starting policy to estimate Flow-Matching residuals that identify moments it fits less accurately. Physical trajectory time t is distinct from Flow-Matching time τ. Let $c_{e,t}$ denote policy conditioning and $a_{e,t} \in \mathbb{R}^{H \times D}$ a normalized expert action chunk, with horizon H and action dimension D. For Gaussian noise ϵ, the linear path is $x_{\tau,e,t} = (1-\tau)\epsilon + \tau a_{e,t}$ and the target velocity is $u_{e,t} = a_{e,t} - \epsilon$. Writing $\xi = (\tau, \epsilon)$, the training loss is

$$\ell_\theta(d, e, t; \xi) = \frac{1}{HD} \left\| v_\theta(x_{\tau,e,t}, \tau, c_{e,t}) - u_{e,t} \right\|_F^2. \tag{2}$$

A frozen copy of the starting policy, θ_0, defines the residual score

$$s_{e,t} = \mathbb{E}_{\xi \sim P_\xi} \left[\ell_{\theta_0}(d, e, t; \xi) \right]. \tag{3}$$

To reduce scoring cost, we estimate this score at $\min(K, N_e)$ evenly spaced valid anchors, with $K \geq 2$, including both endpoints when $N_e > 1$. Linear interpolation along physical time gives a dense estimate $\widehat{s}_{e,t}$.

Temporal proposal. We convert the estimated scores into within-trajectory percentile ranks so that sampling depends on their ordering rather than their numerical gaps. For $N_e > 1$,

$$\rho_{e,t} = \frac{\mathrm{rank}(\widehat{s}_{e,t}) - 1}{N_e - 1}, \qquad r(t \mid e, d) = \frac{\delta + \rho_{e,t}}{\sum_{j \in \mathcal{T}_e} (\delta + \rho_{e,j})}, \quad \delta > 0. \tag{4}$$

Larger residuals receive larger ranks, with average ranks for ties. For $N_e = 1$, set $\rho_{e,t} = 0$ and $r(t \mid e, d) = 1$. Rank calibration therefore defines the residual-guided distribution $r(t \mid e, d)$. A baseline mixture with the baseline temporal distribution then forms the temporal proposal:

$$q(t \mid e, d) = (1 - \alpha) p(t \mid e, d) + \alpha r(t \mid e, d), \qquad 0 \leq \alpha < 1. \tag{5}$$

This mixture ensures $q(t \mid e, d) \geq (1 - \alpha) p(t \mid e, d)$ and therefore retains full support. Uniform sampling is recovered when $\alpha = 0$, all scores are tied, or $N_e = 1$. Offline proposal construction is performed once before continuation training, after which the proposal remains fixed. We use $K = 8$, $\delta = 0.1$, and $\alpha = 0.2$ unless otherwise stated.

Importance correction. Source and trajectory marginals follow the baseline sampling distribution, while anchors are sampled from $q(t \mid e, d)$ and Flow-Matching variables from P_ξ. Because only temporal sampling changes, the importance weight is $w(d, e, t) = p(t \mid e, d)/q(t \mid e, d)$. For a minibatch of B examples, the importance-corrected continuation loss is

$$\widehat{\mathcal{L}}_{\mathrm{ReTime}}(\theta) = \frac{1}{B} \sum_{i=1}^{B} w(d_i, e_i, t_i)\, \ell_\theta(d_i, e_i, t_i; \xi_i). \tag{6}$$

Weights are fixed with respect to θ and applied before batch reduction, without clipping or batch self-normalization. If an anchor uses multiple Flow-Matching draws, its weight multiplies the loss averaged over those draws.

Conditioned on the fixed proposal, for any fixed θ with integrable losses and gradients,

$$\mathbb{E}_{q,\xi}[w \ell_\theta] = \mathbb{E}_{p,\xi}[\ell_\theta] = \mathcal{L}_p(\theta), \qquad \mathbb{E}_{q,\xi}[w \nabla_\theta \ell_\theta] = \mathbb{E}_{p,\xi}[\nabla_\theta \ell_\theta]. \tag{7}$$

These identities follow by cancellation of q with p/q and hold regardless of scoring or interpolation accuracy. Thus, importance correction preserves the baseline continuation objective and its expected

raw gradient. The mixture also bounds the weights by $0 < w \leq (1-\alpha)^{-1}$, giving $w \leq 1.25$ at the default $\alpha = 0.2$; Appendix A.5 provides the derivation.

The temporal proposal can change stochastic-gradient fluctuations while preserving the expected raw gradient. Residuals provide a forward-pass signal for this allocation, but do not guarantee variance reduction or improved continuation training; effectiveness is evaluated empirically (Section 4.3). The identities apply to raw gradients at a common parameter value and do not imply identical optimization paths or expected updates after gradient clipping or Adam preconditioning.

3.3 Efficient Implementation

Offline proposal construction shards demonstration trajectories across eight GPUs. Each device evaluates the frozen starting policy in evaluation mode without backpropagation and batches the selected anchors over the prescribed Flow-Matching-time and noise draws. Expensive policy evaluation is therefore restricted to $K_e = \min(K, N_e)$ anchors per trajectory; linear interpolation, rank calibration, and the baseline mixture are subsequently computed without additional policy evaluations. The resulting per-trajectory temporal proposals are cached once and reused throughout continuation training, eliminating online rescoring and adding no inference-time computation. Appendix A.3 provides implementation details, while Appendix I.3 reports measured training throughput and projected preprocessing cost.

4 Experiments

4.1 Experimental Setup

Benchmarks and metrics. We evaluate GR00T-style policies implemented in StarVLA (Bjorck et al., 2025; StarVLA Community, 2026), with Florence-2 (Xiao et al., 2024) and Qwen3-VL (Bai et al., 2025) backbones. Policies continued on Bridge-family demonstrations (Walke et al., 2023) are evaluated on four WidowX tasks in SimplerEnv (Li et al., 2025), with 96 rollouts per task (384 total). For RoboTwin (Chen et al., 2025), we follow Jing et al. (2026) and evaluate 20 tasks under Easy and Hard conditions, with 25 rollouts per task and condition (500 per condition). Simulation results report binary strict success.

We evaluate on the dual-arm Xtrainer platform using a long-horizon Phone packing task and a fine-grained Bottle picking task. For each task, all methods are evaluated on the same 10 predefined initial object configurations, with one rollout per configuration. Phone is scored by normalized task progress over three sequential stages—pickup, placement into the box, and lid closing—with one point per completed stage. Bottle uses binary success and is counted as successful only when the full task is completed. Additional hardware, task, and evaluation details are provided in Appendix B.7.

Baselines. Uniform retains the baseline sampling distribution and unweighted loss. FrameSkip (Yu et al., 2026a) scores frames using action and visual-temporal information, retains selected frames, and remaps training samples to them. KF-Weight (Wen et al., 2021) assigns a $5\times$ loss weight to the 10% of samples with the largest errors from an action-history predictor, without changing sampling. Loss-IS (Katharopoulos & Fleuret, 2018) resamples examples according to their current training loss and applies inverse-probability correction. Appendix B.4 provides implementation details.

Continuation protocol. Within each experimental setting, all methods share the same starting policy state and use the same training data, optimization configuration, and continuation budget. In simulation, we first train the base policy until the training loss has largely converged, yielding a well-optimized starting policy. We then restore the same complete training state and continue for 500 additional updates to evaluate whether finite-budget continuation can provide further gains beyond an already well-trained policy. The real-world experiments use matched continuation within each setting; Florence–Phone starts from a 20-epoch policy and continues for an additional 20 epochs. Together, these settings cover starting policies at different levels of optimization and assess whether the proposed method remains effective across different initial policy qualities. Available checkpoint and optimization details are provided in Appendix B.

Table 1: **Policy performance across simulation and real-world continuation settings.** Simulation results report strict success (%, ↑); Phone reports normalized three-stage progress and Bottle binary success. Real-World Avg. averages the unrounded Phone and Bottle scores. **Bold** and underline denote the best and second-best results, respectively.

	Method	SimplerEnv (WidowX)					RoboTwin			Real-World		
		Avg.	Stack	Carrot	Spoon	Eggplant	Avg.	Easy	Hard	Avg.	Phone	Bottle
Florence	Starting Policy	36.46	4.17	12.50	44.79	84.38	27.90	27.40	**28.40**	31.67	43.33	20.00
	Uniform	42.19	13.54	32.29	52.08	70.83	20.70	19.40	22.00	50.00	60.00	40.00
	FrameSkip	48.18	14.58	42.71	45.83	**89.58**	28.60	30.20	27.00	48.33	66.67	30.00
	KF-Weight	42.97	13.54	40.63	39.58	78.13	19.50	18.80	20.20	43.33	56.67	30.00
	Loss-IS	48.18	**29.17**	32.29	45.83	85.42	21.80	21.40	22.20	60.00	70.00	50.00
	ReTime	**59.38**	25.00	**46.88**	**86.46**	79.17	**29.00**	**30.40**	27.60	**76.67**	**83.33**	**70.00**
Qwen	Starting Policy	33.85	**10.42**	30.21	23.96	70.83	6.70	7.60	5.80	43.33	46.67	40.00
	Uniform	38.02	**10.42**	27.08	29.17	85.42	8.00	8.00	8.00	58.33	56.67	60.00
	FrameSkip	32.29	8.33	25.00	15.63	80.21	8.20	9.00	7.40	56.67	73.33	40.00
	KF-Weight	38.02	7.29	23.96	**33.33**	87.50	6.50	7.20	5.80	38.33	46.67	30.00
	Loss-IS	39.58	5.21	31.25	**33.33**	88.54	8.70	9.00	8.40	66.67	73.33	60.00
	ReTime	**41.15**	9.38	**35.42**	30.21	**89.58**	**10.20**	**11.20**	**9.20**	**80.00**	**90.00**	**70.00**

4.2 Main Results

Simulation. Table 1 reports continuation-training results across four simulation settings. Additional training does not necessarily improve an already strong policy. On Florence–RoboTwin, for example, the Starting Policy achieves 27.90% strict success, whereas Uniform, KF-Weight, and Loss-IS decrease to 20.70%, 19.50%, and 21.80%, respectively. In contrast, ReTime improves upon the Starting Policy in all four backbone–benchmark settings, showing that its gains are not simply due to additional optimization. ReTime also achieves the highest average strict success among continuation methods in every setting. On Florence–SimplerEnv, it reaches 59.38%, outperforming the strongest baselines, FrameSkip and Loss-IS at 48.18%, by 11.20 percentage points, while remaining best across Qwen–SimplerEnv and both RoboTwin settings. These results indicate more effective use of the limited continuation budget across backbones and benchmarks. Sec. 4.3 uses controlled factorial ablations to disentangle residual-guided temporal allocation, objective preservation, and their interaction; detailed task-level results are provided in Appendix C.

Real-world. On Phone, ReTime achieves normalized task-progress scores of 83.33% with Florence and 90.00% with Qwen, exceeding the strongest continuation baselines by 13.33 and 16.67 percentage points, respectively. On Bottle, ReTime achieves 70.00% success with both backbones, versus 50.00% and 60.00% for the strongest Florence and Qwen baselines. We further extend training on Florence–Phone from the 20-epoch starting checkpoint to 60 epochs, evaluating at 10-epoch intervals. Uniform peaks at 63.3% at 50 epochs before dropping to 56.7% at 60 epochs, whereas ReTime continues to improve to 86.7%, suggesting that it is less susceptible to late-stage performance degradation under prolonged continuation training (Figure 1; see Appendix C.3.1 for detailed results). Protocol and per-rollout Phone results are provided in Appendices B.7 and C.3, respectively.

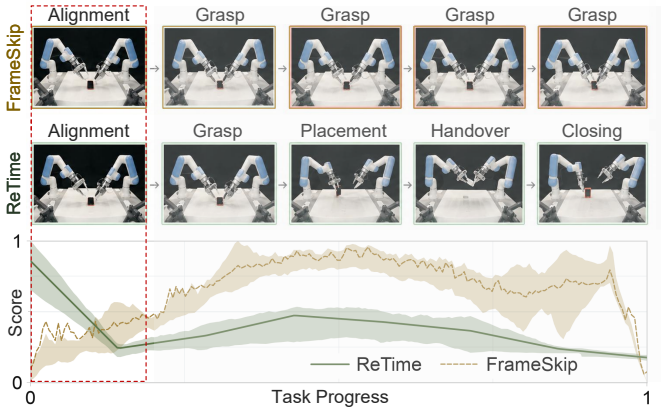


Figure 3: **Qualitative comparison.** The top two rows show representative FrameSkip and ReTime rollouts, respectively; FrameSkip stalls at grasping, while ReTime completes the full task. The bottom panel shows their frame-wise scores aggregated over all episodes along normalized task progress

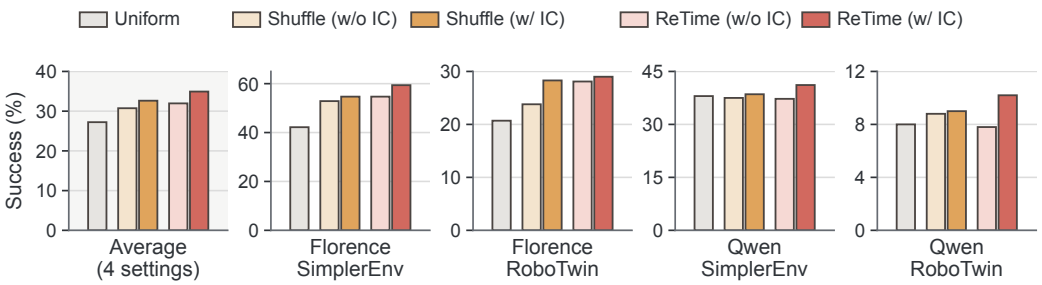


Figure 4: **Temporal allocation and importance correction.** Average strict success across four backbone–benchmark settings after 500 additional updates. Shuffle and ReTime are evaluated with and without importance correction (IC), with Uniform as the reference.

Figure 3 shows a representative qualitative case. FrameSkip identifies salient frames from changes in actions and observations, and therefore assigns higher scores around grasping and other phases with pronounced state changes. In contrast, ReTime residuals peak earlier, during approach and alignment, resulting in nearly opposite temporal emphasis. Because the phone's initial pose varies, the policy must continuously adjust the end-effector pose before grasping. Although this interval contains few abrupt action or observation changes, it can still be a major source of policy error. Consistently, the FrameSkip rollout stalls at the grasp stage, with misalignment already evident beforehand, whereas ReTime allocates more training exposure to this preceding difficult interval. This example suggests that fixed heuristics based on action or observation changes may not align with policy-specific difficulty, whereas frozen-policy residuals provide a task- and policy-dependent temporal signal directly tied to prediction error.

4.3 Mechanistic Analysis

Temporal allocation and residual alignment. Figure 4 crosses temporal allocation with and without importance correction (IC). Uniform samples temporal anchors uniformly. ReTime constructs residual-induced proposal masses and keeps them aligned with their corresponding temporal locations, assigning higher sampling probability to high-residual moments. Shuffle uses the same proposal masses as ReTime but randomly permutes their assignment within each trajectory, preserving their multiset while removing residual–temporal alignment.

Averaged across the four backbone–benchmark settings, the same ordering holds with and without IC: Shuffle improves over Uniform, and ReTime further improves over Shuffle. With IC, Shuffle exceeds Uniform in all four settings, showing that non-uniform temporal allocation can benefit finite-budget continuation even without residual alignment. ReTime further exceeds Shuffle by 0.70–4.69 percentage points, indicating an additional benefit from aligning higher sampling probability with residual-indicated moments. For example, on Florence–SimplerEnv, strict success increases from 42.19% for Uniform to 54.69% for Shuffle and 59.38% for ReTime.

Importance correction. Holding the temporal proposal fixed, importance correction further improves performance under both temporal assignments. Averaged across the four settings, strict success increases from 30.74% to 32.63% for Shuffle, a 6.2% relative improvement, and from 31.96% to 34.93% for ReTime, a 9.3% relative improvement. These results show that the gains from non-uniform temporal allocation do not require replacing the baseline continuation objective with the proposal-induced objective. Instead, under the same temporal proposal, restoring the baseline continuation objective through importance correction preserves these gains and further improves performance. Thus, encounter frequency and objective weight can be controlled separately, directly supporting our explicit decoupling of temporal allocation and objective reweighting. Additional factorial results and permutation-sensitivity analyses are provided in Appendices E and E.3.

Residual-signal diagnostics. Table 2 evaluates whether frozen-policy residuals provide a stable and optimization-relevant signal for temporal allocation. Rank ρ and Top-2 overlap measure the consistency of residual rankings across independent estimates, Grad. ρ measures correlation with action-

Table 2: **Residual diagnostics and training-seed robustness.** (a) Residual rank consistency, gradient correlation, and second-moment ratio on 512 trajectories per backbone. (b) Florence–SimplerEnv strict success across three training seeds after 500 additional updates. Only the second-moment ratio is lower-better; bold denotes the best result.

(a) Residual signal diagnostics

	Reliability		Optimization relevance		
Backbone	Rank ρ	Top-2	Grad. ρ	Improved	2nd mom.
Florence	0.962	0.897	0.653	84.0%	0.971
Qwen	0.960	0.931	0.762	94.7%	0.924

(b) Training-seed robustness

Method	Training seed			Mean
	117465	533389	944964	
Uniform	42.2	43.0	58.6	47.9
Shuffle	54.7	54.4	59.1	56.1
ReTime	**59.4**	**63.3**	**61.5**	**61.4**

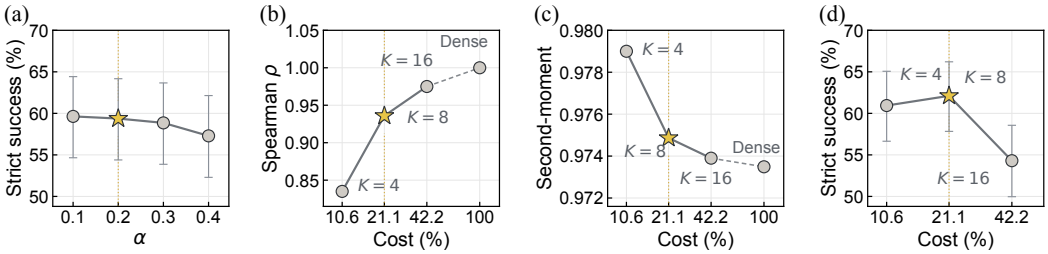


Figure 5: **Guidance strength and scoring resolution on Florence–SimplerEnv.** (a) Strict success versus α. (b) Spearman rank correlation with Dense scoring versus K. (c) Second-moment ratio $M(q)/M(p)$ versus K; lower is better. (d) Strict success versus K. Stars mark defaults; diamonds denote offline-only Dense scoring. Error bars in (a,d) are 95% Wilson intervals over binary rollout outcomes ($n = 384$ and 512 per point, respectively); they do not capture training-seed variability.

gradient magnitude, and the final two columns quantify whether the resulting proposal reduces the second-moment proxy relative to uniform sampling. Residual rankings are highly consistent, with Rank $\rho \approx 0.96$ and Top-2 overlap above 0.89, while residual–gradient correlations reach $\rho = 0.653$ for Florence and 0.762 for Qwen. The residual-guided proposal reduces the second-moment proxy on 84.0% and 94.7% of trajectories, with aggregate ratios of 0.971 and 0.924, respectively. These fixed-panel diagnostics support frozen-policy residuals as a stable, optimization-relevant signal, but do not establish lower full-policy gradient variance or coverage of all four simulation settings. Details on target construction, second-moment estimation, complete statistics, and the frozen-versus-fresh proposal comparison are provided in Appendix F.

Training-seed robustness. On Florence–SimplerEnv, the ordering ReTime > Shuffle > Uniform holds for all three continuation-training seeds (Table 2b). Their strict-success rates are $61.37 \pm 1.95\%$, $56.08 \pm 2.63\%$, and $47.92 \pm 9.25\%$, respectively (mean $\pm$ sample standard deviation). ReTime exceeds Shuffle by 2.34–8.85 percentage points across seeds. This replication supports the alignment advantage within this setting and these three methods. Per-seed results and a paired analysis that retains the training-seed grouping are in Appendices C.2 and E.2.

4.4 Robustness and Sensitivity

We examine sensitivity to the residual-mixture strength α and temporal scoring resolution K on Florence–SimplerEnv. All experiments use the same starting state after 20,000 optimizer updates, continuation-training seed 117465, 500 additional updates, and standard importance correction; the α and K sweeps use fixed evaluation panels of 384 and 512 rollouts per setting, respectively. Figure 5 summarizes both studies. Full statistics are provided in Appendix G.2.

Guidance strength. With $K = 8$, performance is stable for $\alpha \in \{0.1, 0.2, 0.3\}$, achieving 59.64%, 59.38%, and 58.85% strict success, respectively. Increasing α to 0.4 reduces performance to 57.29%, 2.08 percentage points below the default $\alpha = 0.2$. This indicates limited sensitivity

within the moderate range and supports $\alpha = 0.2$ as a practical default, rather than a uniquely optimal choice. Additional results and the corresponding p/q_α weight bounds are provided in Appendix G.1.

Scoring resolution. We compare sparse scoring with $K \in \{4, 8, 16\}$ against Dense scoring on the same 512 training trajectories. Sparse scoring evaluates K evenly spaced anchors per trajectory and interpolates the remaining scores before calibration, whereas Dense evaluates all 19,375 valid anchors. With $K = 8$, ReTime requires only 4,096 residual evaluations (21.1% of the Dense workload), while achieving a mean Spearman correlation of 0.9359 with Dense and recovering 94.81% of its second-moment reduction. However, higher proposal fidelity does not translate monotonically into better downstream performance: $K = 4, 8, 16$ achieve 60.94%, 62.11%, and 54.30% strict success, respectively. Thus, dense temporal scoring is unnecessary for effective continuation; among the tested sparse resolutions, $K = 8$ provides the best observed downstream performance while retaining substantially lower scoring cost than denser alternatives. Dense scoring is used only for offline proposal analysis and is therefore excluded from the downstream comparison.

4.5 Performance–Compute Trade-off

Figure 6 compares method-specific computational overhead with downstream performance on Florence–Phone under the same starting policy, training data, continuation budget, and eight-GPU configuration. The x-axis reports projected additional wall-clock time beyond shared continuation training. ReTime achieves 83.33% task progress with only 0.120 hours of one-time offline computation (0.964 GPU-h). Compared with FrameSkip, ReTime incurs 1.55× more additional compute (0.964 vs. 0.620 GPU-h) but improves task progress by 16.67 percentage points (83.33% vs. 66.67%). In contrast, Loss-IS repeatedly re-scores samples during training, accumulating 207.51 wall-clock hours (1660.09 GPU-h), approximately 1,720× the additional compute of ReTime, while achieving 70.00% task progress. Thus, ReTime retains the low-overhead profile of offline methods while providing the strongest performance–compute trade-off. Appendix H details the timing protocol and compute accounting.

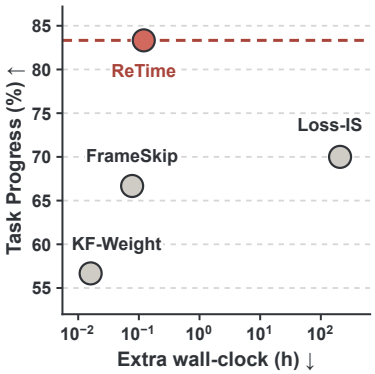


Figure 6: Adaptation Efficiency.

5 Conclusion

We introduce ReTime, which decouples temporal allocation from objective reweighting in continued VLA adaptation by combining residual-guided temporal resampling with importance correction that preserves the baseline continuation objective. Across six simulation and real-world backbone–benchmark settings spanning Florence and Qwen, ReTime achieves the highest average performance, while its advantage persists under longer-horizon Florence–Phone continuation where Uniform eventually degrades. Controlled experiments show that, under a fixed temporal proposal, importance correction consistently improves both temporal assignments, indicating that the gains from non-uniform temporal allocation do not rely on proposal-induced objective reweighting. Moreover, Shuffle outperforms Uniform and ReTime further outperforms Shuffle, supporting separate benefits from non-uniform temporal allocation and residual–time alignment. Overall, our results support explicitly decoupling temporal allocation from objective reweighting, and show that preserving the baseline continuation objective can further improve non-uniform temporal allocation.

Our evaluation spans two backbones, but most settings use one continuation-training run, real-world results use ten fixed configurations per task, and longer-horizon evidence is limited to Florence–Phone; broader training- and scenario-level uncertainty remains to be established. Future work can extend this analysis to broader models, tasks, and training horizons, and examine whether periodically refreshing the residual-guided proposal provides additional downstream gains.

AI USE STATEMENT

We used generative AI tools for manuscript language polishing and consistency checks; they were not used to generate scientific claims or experimental results. We also used them to assist with code development and debugging. The authors reviewed and verified all AI-assisted output and take responsibility for the final paper and code.

REPRODUCIBILITY STATEMENT

We summarize the available information for reproducing the reported results throughout the paper and appendices. Appendix B specifies the available backbone–benchmark configurations, shared starting checkpoints, continuation contracts, comparison-method implementations, and evaluation protocols. Appendix A documents the offline residual scoring, temporal proposal, and importance-corrected loss used by ReTime. Detailed integer rollout outcomes and supporting analyses are reported in the corresponding results appendices. Appendix I summarizes the reproduction path, result aggregation and rounding procedure, computational-cost measurement, and the planned release of the ReTime method implementation as a direct StarVLA integration.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 41–48, 2009. doi: 10.1145/1553374.1553380.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, et al. π_0: A vision-language-action flow model for general robot control, 2024. URL https://arxiv.org/abs/2410.24164.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization. In *Proceedings of the Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 2025.

Anthony Brohan, Noah Brown, Justice Carbajal, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control, 2023. URL https://arxiv.org/abs/2307.15818.

Haw-Shiuan Chang, Erik Learned-Miller, and Andrew McCallum. Active bias: Training more accurate neural networks by emphasizing high variance samples. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/2f37d10131f2a483a8dd005b3d14b0d9-Abstract.html.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan-ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025. URL https://arxiv.org/abs/2506.18088.

Ziyang Chen, Shaoguang Wang, Weiyu Guo, Qianyi Cai, He Zhang, Pengteng Li, Yiren Zhao, and Yandong Guo. PHASER: Phase-aware and semantic experience replay for vision-language-action models. *arXiv preprint arXiv:2606.03598*, 2026. URL https://arxiv.org/abs/2606.03598.

Lingxiao Guo, Zhengrong Xue, Zijing Xu, and Huazhe Xu. DemoSpeedup: Accelerating visuomotor policies via entropy-guided demonstration acceleration. In *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 599–609. PMLR, 2025. URL https://proceedings.mlr.press/v305/guo25a.html.

Asher J. Hancock, Xindi Wu, Lihan Zha, Olga Russakovsky, and Anirudha Majumdar. Actions as language: Fine-tuning vlms into vlas without catastrophic forgetting. In *International Conference on Learning Representations*, 2026.

Yuhua Jiang, Junjie Lu, Xinyao Qin, Xiaoyu Chen, Kaixin Wang, Feifei Gao, and Li Zhao. VLA-GSE: Boosting parameter-efficient fine-tuning in vla with generalized and specialized experts. *arXiv preprint arXiv:2605.06175*, 2026.

Dong Jing, Gang Wang, Jiaqi Liu, Weiliang Tang, Zelong Sun, Yunchao Yao, Zhenyu Wei, Yunhui Liu, Zhiwu Lu, and Mingyu Ding. Mixture of horizons in action chunking. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2511.19433.

Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2525–2534. PMLR, 2018. URL https://proceedings.mlr.press/v80/katharopoulos18a.html.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. In *Robotics: Science and Systems*, 2025a. URL https://roboticsconference.org/2025/program/papers/17/.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2025b. URL https://proceedings.mlr.press/v270/kim25c.html.

M. Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, volume 23, 2010. URL https://proceedings.neurips.cc/paper/2010/hash/e57c6b956a6521b28495f2886ca0977a-Abstract.html.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Oier Mees, Karl Pertsch, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 3705–3728. PMLR, 2025. URL https://proceedings.mlr.press/v270/li25c.html.

Mingyu Liu, Zeju Li, Jiuhe Shu, Hanqing Wang, Yuhao Chao, Hao Chen, and Chunhua Shen. Perfect demo makes poor teacher: Learning robust alignment from critical motion segments. *arXiv preprint arXiv:2606.15587*, 2026a. URL https://arxiv.org/abs/2606.15587.

Yuan Liu, Haoran Li, Shuai Tian, Yuxing Qin, Yuhui Chen, Yupeng Zheng, Yongzhen Huang, and Dongbin Zhao. Towards long-lived robots: Continual learning vla models via reinforcement fine-tuning. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2026b. doi: 10.15607/RSS.2026.XXII.086.

Deanna Needell, Nathan Srebro, and Rachel Ward. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *Advances in Neural Information Processing Systems*, volume 27, pp. 1017–1025, 2014. URL https://proceedings.neurips.cc/paper/2014/hash/b3310bba2be31e673a7ded3386994599-Abstract.html.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8162–8171. PMLR, 2021. URL https://proceedings.mlr.press/v139/nichol21a.html.

Physical Intelligence. $\pi_{0.7}$: A steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, Danny Driess, Michael Equi, Adnan Esmail, Yunhao Fang, Chelsea Finn, Catherine Glossop, Thomas Godden, Ivan Goryachev, Lachy Groom, Hunter Hancock, Karol Hausman, Gashon Hussein, Brian Ichter, Szymon Jakubczak, Rowan Jen, Tim Jones, Ben Katz, Liyiming Ke, Chandra Kuchi, Marinda Lamb, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Yao Lu, Vishnu Mano, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Charvi Sharma, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Will Stoeckle, Alex Swerdlow, James Tanner, Marcel Torne, Quan Vuong, Anna Walling, Haohuan Wang, Blake Williams, Sukwon Yoo, Lili Yu, Ury Zhilinsky, and Zhiyuan Zhou. $\pi^*_{0.6}$: A vla that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1511.05952.

Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 761–769, 2016. URL https://openaccess.thecvf.com/content_cvpr_2016/html/Shrivastava_Training_Region-Based_Object_CVPR_2016_paper.html.

StarVLA Community. Starvla: A lego-like codebase for vision-language-action model developing. *arXiv preprint arXiv:2604.05014*, 2026.

Homer Rich Walke, Kevin Black, Tony Z. Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, Abraham Lee, Kuan Fang, Chelsea Finn, and Sergey Levine. BridgeData V2: A dataset for robot learning at scale. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 1723–1736. PMLR, 2023. URL https://proceedings.mlr.press/v229/walke23a.html.

Chuan Wen, Jierui Lin, Jianing Qian, Yang Gao, and Dinesh Jayaraman. Keyframe-focused visual imitation learning. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 11123–11133. PMLR, 2021. URL https://proceedings.mlr.press/v139/wen21d.html.

Changti Wu, Bin Yu, Zhaolong Shen, Shijie Lian, Xiaopeng Lin, Cong Huang, Zhirui Zhang, Lei Zhang, and Kai Chen. SIEVE: Structure-aware data selection for imitation learning with VLA models. *arXiv preprint arXiv:2607.06442*, 2026. URL https://arxiv.org/abs/2607.06442.

Bin Xiao, Haiping Wu, Weijian Xu, Xiyang Dai, Houdong Hu, Yumao Lu, Michael Zeng, Ce Liu, and Lu Yuan. Florence-2: Advancing a unified representation for a variety of vision tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4818–4829, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Xiao_Florence-2_Advancing_a_Unified_Representation_for_a_Variety_of_Vision_CVPR_2024_paper.html.

Bin Yu, Shijie Lian, Xiaopeng Lin, Zhaolong Shen, Yuliang Wei, Changti Wu, Hang Yuan, Haishan Liu, Bailing Wang, Cong Huang, and Kai Chen. FrameSkip: Learning from fewer but more informative frames in VLA training. *arXiv preprint arXiv:2605.13757*, 2026a. URL https://arxiv.org/abs/2605.13757.

Hanyang Yu, Haitao Lin, Jingbo Zhang, Wenyao Zhang, Chenghao Gu, Heng Li, and Ping Tan. MaskWAM: Unifying mask prompting and prediction for world-action models. *arXiv preprint arXiv:2606.13515*, 2026b. URL https://arxiv.org/abs/2606.13515v1.

Xinglei Yu, Zhenyang Liu, Shufeng Nan, Simo Wu, and Yanwei Fu. VADF: Vision-adaptive diffusion policy framework for efficient robotic manipulation. *arXiv preprint arXiv:2604.15938*, 2026c. URL https://arxiv.org/abs/2604.15938.

Yu Zhang, Yuqi Xie, Huihan Liu, Rutav Shah, Michael Wan, Linxi Fan, and Yuke Zhu. SCIZOR: A self-supervised approach to data curation for large-scale imitation learning. In *IEEE International Conference on Robotics and Automation*, 2026. URL https://arxiv.org/abs/2505.22626. Oral presentation.

A Additional Method Details and Objective Preservation

This part specifies the offline residual scoring and offline proposal construction used by ReTime, and derives the objective-preservation identities stated in Section 3. The action horizon H and action dimension D follow the backbone–benchmark configuration; the construction below applies without changing either quantity.

A.1 Frozen Flow-Matching Residual Estimation

For a valid physical-time anchor, let $a \in \mathbb{R}^{H\times D}$ denote the normalized expert action chunk and let c denote the corresponding policy conditioning. We use the same Flow-Matching path and velocity target as the training loss in Equation 2. Following the policy's Flow-Matching-time distribution, we use the corresponding Beta law

$$b \sim \mathrm{Beta}(1.5, 1), \qquad \tau = \frac{s - \min(b, s)}{s}, \qquad s = 0.999, \tag{8}$$

and pair each time value with Gaussian noise $\epsilon \sim \mathcal{N}(0, I)$. The noisy action and target velocity are

$$x_\tau = (1-\tau)\epsilon + \tau a, \qquad u = a - \epsilon. \tag{9}$$

Sparse residual scoring uses a frozen copy of the starting policy θ_0 in evaluation mode. We approximate the expectation in Equation 3 with $R = 4$ independent estimator replicates and $J = 8$ samples per replicate:

$$\widehat{s}_{e,t}^{(r)} = \frac{1}{J}\sum_{j=1}^{J} \ell_{\theta_0}(d, e, t; \tau_j, \epsilon_j^{(r)}), \qquad \widehat{s}_{e,t} = \frac{1}{R}\sum_{r=1}^{R} \widehat{s}_{e,t}^{(r)}. \tag{10}$$

Each replicate evaluates four midpoint quantiles of the Beta distribution in Equation 8; every quantile is paired with antithetic Gaussian draws z and $-z$. Anchors evaluated in the same scoring batch share the Flow-Matching-time values and antithetic noise draws, so their score ordering is not driven by different Monte Carlo samples. The squared velocity residual is averaged over the action horizon and dimensions before the eight samples and four replicates are averaged. The four quantile levels are $1/8$, $3/8$, $5/8$, and $7/8$; the corresponding Beta quantiles are transformed into Flow-Matching times by Equation 8. These deterministic Flow-Matching-time points approximate the expectation over τ, while antithetic Gaussian sampling approximates the noise expectation. The resulting residual score is an approximation used to construct the temporal proposal; objective preservation depends on the actual sampling probabilities and their importance weights, not on this approximation being exact.

A.2 Sparse Residual Scoring and Offline Proposal Construction

Let a trajectory contain N_e valid physical-time anchors. We evaluate the frozen residual at $K_e = \min(K, N_e)$ positions. For $K_e > 1$, their zero-based indices are

$$i_m = \left\lfloor \frac{m(N_e - 1)}{K_e - 1} \right\rfloor, \qquad m = 0, \ldots, K_e - 1, \tag{11}$$

which covers both endpoints. When $N_e = 1$, the only valid anchor is selected. Linear interpolation along physical time assigns a score to every unscored valid anchor.

Rank calibration converts the interpolated scores to average within-trajectory ranks, so tied scores receive the same rank. For $N_e > 1$, we define the residual-guided distribution

$$\rho_{e,t} = \frac{\mathrm{rank}(\widehat{s}_{e,t}) - 1}{N_e - 1}, \qquad r(t \mid e, d) = \frac{\delta + \rho_{e,t}}{\sum_{j\in\mathcal{T}_e}(\delta + \rho_{e,j})}. \tag{12}$$

For a single-anchor trajectory, we set $\rho_{e,t} = 0$ and $r(t \mid e, d) = 1$. The rank offset $\delta > 0$ assigns positive mass to every valid anchor. A baseline mixture with the baseline temporal distribution then forms the temporal proposal:

$$q(t \mid e, d) = (1-\alpha)p(t \mid e, d) + \alpha r(t \mid e, d), \qquad p(t \mid e, d) = \frac{1}{N_e}. \tag{13}$$

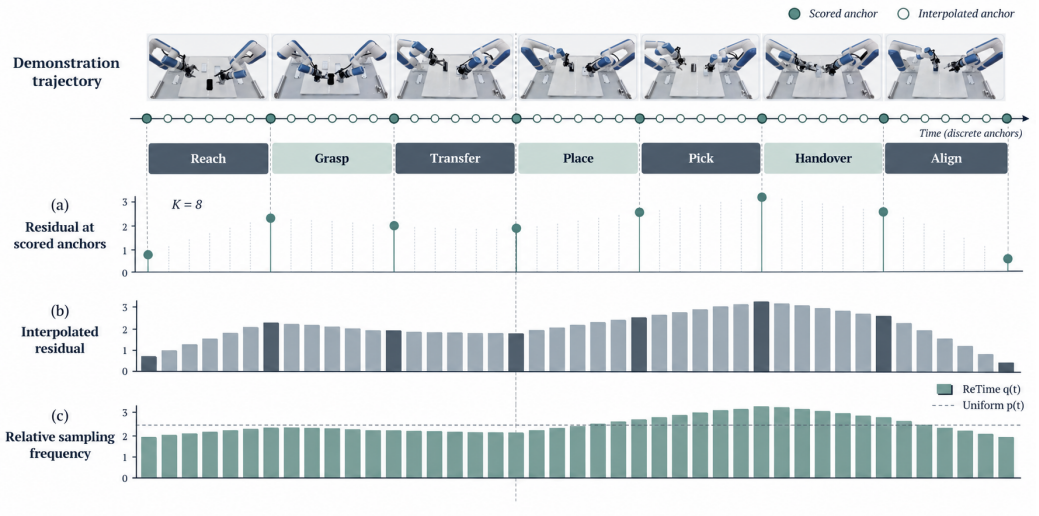


Figure 7: **Sparse residual scoring and offline proposal construction.** A frozen copy of the starting policy scores $K = 8$ sparsely spaced valid anchors, and linear interpolation assigns residual scores to the remaining valid anchors. Rank calibration defines the residual-guided distribution, which is mixed with the baseline temporal distribution to form the full-support temporal proposal. The bottom panel shows the resulting relative sampling frequency with respect to the baseline temporal distribution: values above one correspond to moments sampled more frequently, while values below one correspond to moments sampled less frequently.

Unless otherwise stated, we use $K = 8$, $\delta = 0.1$, and $\alpha = 0.2$. Offline proposal construction is performed once using the frozen starting policy, and the temporal proposal remains fixed during continuation training. Dataset and trajectory marginals are inherited from the baseline sampling distribution; only the temporal conditional used for sampling differs from the baseline temporal distribution in Equation 13.

Figure 7 illustrates this construction along a demonstration trajectory. During offline residual scoring, a frozen copy of the starting policy evaluates $K = 8$ sparse physical-time anchors. Linear interpolation assigns residual scores to all valid anchors, rank calibration defines the residual-guided distribution, and the baseline mixture forms the temporal proposal. The resulting proposal reallocates sampling toward high-residual moments while retaining support over the full trajectory.

A.3 Parallel Scoring and Proposal Reuse

Parallel offline residual scoring. The offline residual-scoring workload is partitioned by demonstration trajectory across eight GPUs. Each device evaluates the frozen starting policy in evaluation mode on its assigned trajectories, without backpropagation. Selected anchors and their Flow-Matching-time/noise evaluations are batched within each device. The estimator follows Appendix A.1, including the shared Flow-Matching-time points and antithetic noise within each scoring batch.

Scoring workload. With $R = 4$ replicates and $J = 8$ Flow-Matching-time/noise evaluations per replicate, each selected anchor contributes 32 loss evaluations. A trajectory with $K_e = \min(8, N_e)$ selected anchors therefore contributes $32K_e \leq 256$ evaluations. These counts describe anchor–Flow-Matching-time–noise combinations, which can be processed together in a batch. After scoring the selected anchors, linear interpolation, rank calibration, and baseline mixture are performed within each trajectory according to Appendix A.2.

Proposal reuse. The offline cache supplies a fixed temporal proposal over each trajectory's valid physical-time anchors. During continuation training, source and trajectory selection follow the baseline sampling distribution, while temporal anchors follow this proposal. The importance weight is computed from the probability used to sample the selected anchor. The same proposal is reused

throughout continuation training, so the continuation policy is not rescored as its parameters change. The offline cache and residual scorer are not required at inference.

Cost accounting. The preprocessing workload depends on the shared starting checkpoint, dataset, and scoring configuration. Its one-time cost is separate from continuation training. Appendix I.3 reports training-update timings and projected ReTime cache costs; these estimates exclude data I/O and do not measure end-to-end deployment latency.

A.4 Importance-Corrected Continuation

For an anchor sampled from $q(t \mid e, d)$, importance-corrected continuation uses the ordinary importance weight $w(d, e, t) = p(t \mid e, d)/q(t \mid e, d)$ derived in Equation 17. For a minibatch of B examples, we optimize the importance-corrected loss

$$\widehat{\mathcal{L}}_{\mathrm{ReTime}}(\theta) = \frac{1}{B} \sum_{i=1}^{B} w(d_i, e_i, t_i)\, \ell_\theta(d_i, e_i, t_i; \xi_i). \tag{14}$$

The anchor weight is applied before batch reduction and is not normalized by the sum of weights in the minibatch. If multiple Flow-Matching draws are used for one training example, their losses are averaged first and then multiplied by the anchor weight.

A.5 Objective Preservation and Gradient Fluctuations

Objective preservation. Condition on the temporal proposal constructed before continuation training. The baseline sampling distribution and proposal-induced sampling distribution are

$$p(d, e, t) = p(d)p(e \mid d)p(t \mid e, d), \tag{15}$$
$$q(d, e, t) = p(d)p(e \mid d)q(t \mid e, d). \tag{16}$$

The source and trajectory distributions and the Flow-Matching distribution P_ξ are unchanged. On the support of p, the importance weight is therefore

$$w(d, e, t) = \frac{p(d, e, t)}{q(d, e, t)} = \frac{p(t \mid e, d)}{q(t \mid e, d)}. \tag{17}$$

For any fixed θ and any integrable scalar- or vector-valued function $f_\theta(d, e, t; \xi)$,

$$\begin{aligned}\mathbb{E}_{q,\xi}[w f_\theta] &= \sum_{d,e} p(d)p(e \mid d) \sum_{t \in \mathcal{T}_e} q(t \mid e, d) \frac{p(t \mid e, d)}{q(t \mid e, d)} \mathbb{E}_\xi[f_\theta(d, e, t; \xi)] \\ &= \sum_{d,e} p(d)p(e \mid d) \sum_{t \in \mathcal{T}_e} p(t \mid e, d) \mathbb{E}_\xi[f_\theta(d, e, t; \xi)] = \mathbb{E}_{p,\xi}[f_\theta].\end{aligned}$$

Taking $f_\theta = \ell_\theta$ or $f_\theta = \nabla_\theta \ell_\theta$ gives the identities for the baseline objective and expected raw gradient in Equation 7. Identifying the expected raw gradient with $\nabla_\theta \mathcal{L}_p$ additionally requires the usual conditions for interchanging differentiation and expectation. The identity depends on using the actual temporal-proposal probability in the ratio, without clipping or self-normalizing the importance weights.

By Equation 5,

$$q(t \mid e, d) \geq (1 - \alpha)p(t \mid e, d) > 0 \tag{18}$$

on the baseline support, so

$$0 < w(d, e, t) \leq \frac{1}{1 - \alpha}. \tag{19}$$

These statements hold conditional on any realized proposal satisfying the support condition. Approximation errors in scoring or interpolation can affect sampling efficiency, but do not alter these identities.

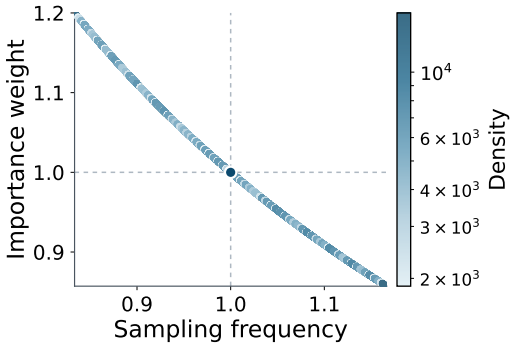


Figure 8: **Objective preservation via importance correction.** For the Florence–SimplerEnv ReTime temporal proposal ($K = 8$, $\alpha = 0.2$), each moment's relative sampling frequency q/p is paired with the reciprocal importance weight p/q. The empirical density follows this reciprocal relation; the marked point $(1, 1)$ denotes unchanged sampling and unit weight.

Gradient fluctuations. At a fixed parameter value θ and for a fixed trajectory (d, e), write $p_t = p(t \mid e, d)$ and $q_t = q(t \mid e, d)$. Let G_t be the random per-anchor gradient, including any averaging over multiple Flow-Matching draws, and let $\mu_e = \sum_t p_t \mathbb{E}[G_t]$. For $T \sim q(\cdot \mid e, d)$, the corrected gradient $Z = (p_T/q_T)G_T$ satisfies $\mathbb{E}[Z] = \mu_e$. Assuming finite second moments,

$$M_e(q; \theta) = \mathbb{E}[\|Z\|_2^2] = \sum_{t \in \mathcal{T}_e} \frac{p_t^2}{q_t} \mathbb{E}[\|G_t\|_2^2]. \tag{20}$$

Thus $\operatorname{tr} \operatorname{Cov}(Z) = M_e(q; \theta) - \|\mu_e\|_2^2$: changing q can change the gradient fluctuations while preserving the expected raw gradient. The proposed residual score does not establish that $M_e(q; \theta)$ is smaller than its baseline value. Gradient magnitudes also depend on the model's parameter sensitivity, and the temporal proposal is constructed once using the frozen starting policy. Any improvement in finite-budget continuation training must therefore be supported empirically.

Figure 8 visualizes the importance correction over all valid Florence–SimplerEnv physical-time anchors. ReTime changes the relative sampling frequency of each moment by $q(t \mid e, d)/p(t \mid e, d)$ and applies the reciprocal importance weight $p(t \mid e, d)/q(t \mid e, d)$. Their pointwise product is one, illustrating how temporal exposure changes while the baseline continuation objective and expected raw gradient are preserved.

A.6 Shuffle Proposal Construction

The Shuffle control begins with the calibrated residual masses in Equation 12. Within each trajectory, it samples an independent permutation π_e over the complete valid-anchor sequence and assigns

$$r_{\text{shuf}}(t \mid e, d) = r(\pi_e(t) \mid e, d), \qquad q_{\text{shuf}}(t \mid e, d) = (1 - \alpha)p(t \mid e, d) + \alpha r_{\text{shuf}}(t \mid e, d). \tag{21}$$

Fixed points are allowed under the permutation. Because the baseline temporal distribution is uniform within each trajectory, this construction preserves exactly the per-trajectory multisets of calibrated masses, proposal probabilities, and importance weights, while removing their residual-guided assignment to physical time. When importance correction is enabled, its weights are computed directly from q_{shuf} using Equation 17.

B Experimental Configurations and Evaluation Protocols

This appendix specifies the policy configurations, continuation contracts, comparison-method implementations, and evaluation panels used in the main experiments. Table 3 summarizes the four backbone–benchmark settings and the checkpoints reported in the main comparison.

Table 3: Backbone–benchmark configurations for the main simulation comparison. Every setting restores the complete shared training state and continues for 500 optimizer updates. Bridge uses one view and 16×7 chunks; RoboTwin uses three views and 50×14 chunks.

Setting	Backbone	Training data	Optimizer updates	Global batch
Florence–SimplerEnv	Florence-2-large	Bridge	20,000 → 20,500	64
Qwen–SimplerEnv	Qwen3-VL-4B	Bridge	29,000 → 29,500	48
Florence–RoboTwin	Florence-2-large	RoboTwin	10,000 → 10,500	64
Qwen–RoboTwin	Qwen3-VL-4B	RoboTwin	11,000 → 11,500	40

Table 4: Optimization contracts for the four simulation settings. Learning rates are base/VLM interface/action head. Every schedule is cosine with the listed number of warmup optimizer updates, then restored. Batch geometry is per-device batch × devices × gradient accumulation.

Setting	Optimizer	Learning rates	Warmup updates	Batch
Florence–SimplerEnv	AdamW	$3\times10^{-5}/10^{-5}/10^{-4}$	5,000	$4\times2\times8$
Qwen–SimplerEnv	8-bit AdamW	$3\times10^{-5}/10^{-5}/10^{-4}$	10,000	$2\times1\times24$
Florence–RoboTwin	AdamW	$10^{-5}/10^{-5}/10^{-4}$	5,000	$2\times2\times16$
Qwen–RoboTwin	8-bit AdamW	$10^{-5}/10^{-5}/10^{-4}$	5,000	$5\times2\times4$

B.1 Policy and Action Configurations

The Bridge policies use one 224×224 RGB view and predict normalized 16×7 action chunks. The RoboTwin policies use three synchronized 224×224 RGB views and predict 50×14 absolute joint-position chunks without robot-state input. In all four settings, a DiT-B Flow-Matching action head is conditioned on either Florence-2-large or Qwen3-VL-4B-Instruct features. The VLM interface and action head of the continuation policy remain trainable during continuation training, and policy inference uses four integration steps. The offline residual scoring and temporal-proposal parameters used by ReTime are specified in Appendix A.

B.2 Continuation Protocol

Within each simulation backbone–benchmark setting, all methods resume from the complete shared starting checkpoint, restoring the model, optimizer, learning-rate scheduler, distributed random-number state, sampler and data order, and completed-update counter. They use the same candidate training-data pool, backward-batch size, 500-update horizon, and evaluation panel. Method-specific scoring, selection, remapping, and loss-weighting operations do not change the number of backward optimizer updates.

B.3 Optimization Settings

Table 4 reports the optimizer, learning rates, scheduler, and batch geometry for each setting. Learning rates are listed for the base model, VLM interface, and action head, respectively. All four simulation configurations use Adam betas $(0.9, 0.95)$, epsilon 10^{-8}, weight decay 10^{-8}, gradient clipping at 1.0, and BF16 training.

B.4 Comparison-Method Implementations

Table 5 distinguishes the unit scored and the training intervention made by each method. The implementation details below define the abbreviated method names used in the main table.

Uniform. Uniform retains the baseline sampling distribution and minimizes the unweighted mean Flow-Matching loss. It introduces neither an auxiliary scoring stage nor a change to the physical-time mapping.

Table 5: Operational comparison of continuation methods. AVI/VAC/TPI: action variation, visual–action coherence, and temporal progress. The last column is the loss weight and the scoring schedule.

Method	Score; unit	Intervention	Weight; when
Uniform	none; example	unweighted baseline	none
FrameSkip	AVI/VAC/TPI; frame	retain frames, remap time	none; offline
KF-Weight	action error; example	fixed sampling; loss $1\times$ or $5\times$	none; offline
Loss-IS	current loss; global candidate	resample $B \to b$	$1/(Bg_i)$; each update
ReTime	frozen FM residual; anchor	sample $q(t \mid e, d)$	p/q; once

FrameSkip. Our FrameSkip-adapted implementation follows the informative-frame retention and temporal-remapping procedure of FrameSkip (Yu et al., 2026a). It uses a retention ratio of 0.2, with at least five retained frames per trajectory, and combines action variation, visual–action coherence, and temporal-progress scores with weights $0.6/0.2/0.2$. Visual–action coherence is computed offline with DINOv2-S/14 features from at most 16 frames; this auxiliary model does not replace the policy's visual encoder. Temporal progress uses the public Gaussian fallback with variance 0.2. First and last frames, gripper transitions, and frames above the 90th percentile of action change enter the key-stage candidate set; when this set exceeds the retention budget, the combined importance score determines the retained subset.

During continuation training, five remapped global microbatches are followed by one full-frame microbatch. A logical time index in a remapped batch is assigned to the first retained physical time not earlier than that index, with indices beyond the retained support clipped to its final element. On RoboTwin, the selected physical time is shared by all three camera views. FrameSkip uses its own retention schedule and does not apply ReTime's p/q importance correction.

KF-Weight. The KF-Weight row implements the step-weighting variant of Keyframe-Focused Imitation Learning (Wen et al., 2021). An action-history-only predictor uses the previous nine actions to predict the current action. It has one hidden layer of width 300 and a leaky ReLU with negative slope 0.2, and is trained for 300 epochs using Adam with learning rate 10^{-3}, betas $(0.9, 0.999)$, and batch size 512. Predictors and 90th-percentile action-prediction-error thresholds are fit separately for each data source. Errors are computed with equal-weight MSE in the normalized action space, using seven action dimensions on Bridge and 14 on RoboTwin.

Sampling remains unchanged. The complete per-example Flow-Matching chunk loss receives weight 5 for the highest-error 10% of examples and weight 1 otherwise, and the weighted losses are averaged over the backward batch. This baseline does not construct a temporal proposal or apply p/q importance correction.

Loss-IS. Loss-IS adapts online example-level importance sampling (Katharopoulos & Fleuret, 2018). At every optimizer update, it first draws a candidate pool of size $B = 3b$ from its baseline training distribution, where b is the backward-batch size. The continuation policy evaluates a fresh per-example training loss s_i for each candidate and forms

$$g_i = \frac{s_i}{\sum_{j=1}^{B} s_j}. \tag{22}$$

It then draws b examples with replacement from this candidate pool and weights a selected example by $1/(Bg_i)$. Loss-IS recomputes these scores at every update; it does not reuse the frozen ReTime residual cache or construct a persistent within-trajectory temporal proposal.

B.5 Bridge Evaluation in SimplerEnv

Both Bridge policies are evaluated on the WidowX embodiment in SimplerEnv (Li et al., 2025; Walke et al., 2023). The fixed panel contains Stack, Carrot, Spoon, and Eggplant. Each task contributes 96 rollouts and each method–checkpoint pair contributes 384 rollouts. We report the simulator's binary strict-success outcome and aggregate by summing successes over the fixed panel.

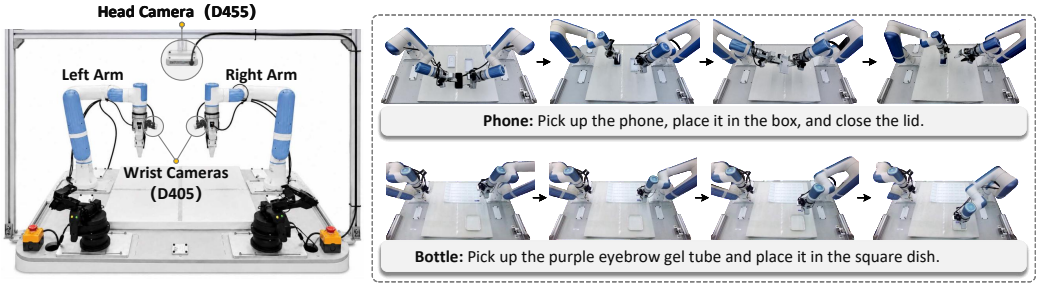


Figure 9: **Real-world evaluation setup and task execution.** The shared dual-arm Xtrainer platform is shown on the left. Representative Phone and Bottle execution stages are shown in the upper and lower rows, respectively.

Evaluation uses a 5-Hz policy control rate, a 500-Hz simulator rate, at most 120 policy steps, and action scale 1.0. The same task definitions, rollout counts, and success criterion are used for Florence–SimplerEnv and Qwen–SimplerEnv.

B.6 RoboTwin Evaluation

Like Jing et al. (2026), we evaluate a fixed RoboTwin task subset rather than the full benchmark. Our panel contains 20 tasks, whereas their study used seven; the same 20 tasks are used for both backbones and all continuation methods. Task names and per-task outcomes appear in Tables 7 and 8.

Each checkpoint is evaluated under Easy and Hard conditions (Chen et al., 2025). We collect 25 valid rollouts for every task and condition, yielding 500 rollouts per condition and 1000 per checkpoint. Easy and Hard rates are computed over their respective 500-rollout panels; the overall rate pools all 1000 binary task successes. The task set, condition definitions, rollout budget, and aggregation are identical for Florence–RoboTwin and Qwen–RoboTwin.

B.7 Real-World Experimental Setup and Evaluation

B.7.1 Hardware Platform

We conduct the physical evaluation on the dual-arm Xtrainer platform, also used in MaskWAM (Yu et al., 2026b). The platform has two robotic arms operating over a shared tabletop workspace. The policy receives three synchronized RGB streams—a head-mounted RealSense D455 overview and left- and right-wrist RealSense D405 views—together with proprioceptive robot state. Actions are absolute bimanual end-effector poses rather than joint commands. Each arm is represented as position, orientation, and a continuous gripper opening; at training time each quaternion is converted to rotation-6D, so the model predicts 16-step chunks of dimension 20, while the robot controller receives the equivalent 16-dimensional xyzw command. Control runs at 10 Hz.

B.7.2 Task Setups and Execution

Figure 9 shows the platform and execution stages. The main results are reported in Table 1.

Phone task. The Phone task is a bimanual three-stage sequence: picking up a phone, placing it inside a box, and closing the box lid. Completing the task requires the policy to carry the manipulation sequence through the final lid-closing stage. Training and evaluation use the English instruction "Pick up the phone, place it in the box, and close the lid." The training corpus contains 305 demonstrations (111,892 frames). We record completion of each stage so that the evaluation captures both intermediate progress and successful execution of the entire sequence.

Bottle task. The Bottle task is a primarily single-arm two-stage transfer: the robot picks a bottle from a source tray and places it in a target tray. The training corpus contains 400 demonstrations (75,758 frames) and four object-specific English instructions, which we keep distinct at evaluation:

Table 6: Training configuration for the Florence–Phone horizon experiment. Both continuations use AdamW, $\beta = (0.9, 0.95)$, $\epsilon = 10^{-8}$, weight decay 10^{-8}, VLM learning rate 10^{-5}, action-head learning rate 10^{-4}, a cosine schedule with floor 5×10^{-7} and 5,000 warmup updates, global batch $4 \times 8 \times 4 = 128$, and 873 updates per epoch. The epoch-20 Uniform state is the common starting state for the continuation comparison.

Field	Uniform	ReTime
Initialization	From epoch 0	Restored epoch 20
Start update	17,460	17,460
Final update	52,380	52,380
Final epoch	60	60
Seed	42	117465
Restored state	None	Model, optimizer, scheduler, and RNG

"Pick up the pink primer bottle and place it in the square dish"; "Pick up the pink dropper and place it in the square dish"; "Pick up the purple eyebrow gel tube and place it in the square dish"; and "Pick up the silver eyebrow gel tube and place it in the square dish." The ten evaluation rollouts use these four objects in a 3–3–2–2 split (pink primer and pink dropper three times each; purple and silver eyebrow-gel tubes twice each). We record completion of the pickup and placement stages so that the evaluation captures both partial progress and successful completion of the full transfer.

B.7.3 Real-World Training Configuration Coverage

Table 6 records the available training configuration for the Florence–Phone horizon experiment. Uniform is trained continuously from epoch 0. Its epoch-20 state after 17,460 optimizer updates is the common starting point for the continuation comparison; ReTime restores that complete state, while Uniform training continues uninterrupted. All compared methods receive the same number of post-epoch-20 updates. The documented Uniform and ReTime continuations both use 873 optimizer updates per epoch and reach 52,380 optimizer updates (epoch 60).

B.7.4 Evaluation Panel and Metrics

The reported evaluation contains ten physical rollouts for each method with each of the Florence and Qwen backbones. The comparison includes the starting policy, Uniform, FrameSkip, KF-Weight, Loss-IS, and ReTime. The Florence–Phone main comparison uses the shared epoch-20 state and evaluates each method after the same 20-epoch continuation budget at total epoch 40 in Section 4.2. Uniform training continues uninterrupted from that state, whereas the other methods restore it before continuation. Both backbones are evaluated under the same physical trial protocol.

For each task, the evaluation panel specifies ten predefined initial object-and-target configurations, including different target positions. Every method is tested once on each of the same ten configurations. Before a trial, objects and workspace fixtures are reset to that configuration's designated positions, without additional randomized offsets; thus the layout is fixed within a configuration, not identical across all ten trials. The policy then runs without human intervention until 600 control steps have elapsed (60 s at 10 Hz). Stage completion is judged by visual inspection of the recorded rollout; we do not apply a geometric distance threshold. A stage is credited only if it is completed during that trial.

For the Phone task, let $x_{i,1}, x_{i,2}, x_{i,3} \in \{0, 1\}$ indicate completion of phone pickup, placement inside the box, and lid closing, respectively. Each completed stage contributes one point, giving

$$s_i^{\text{Phone}} = x_{i,1} + x_{i,2} + x_{i,3} \in \{0, 1, 2, 3\}. \tag{23}$$

For $N = 10$ rollouts, Phone normalized stage progress and Bottle binary full-task success are

$$\text{Phone stage progress} = \frac{100}{3N} \sum_{i=1}^{N} s_i^{\text{Phone}}, \tag{24}$$

$$\text{Bottle full-task success} = \frac{100}{N} \sum_{i=1}^{N} \mathbf{1}[y_{i,1} = 1 \ \wedge \ y_{i,2} = 1], \tag{25}$$

Here $y_{i,1}$ and $y_{i,2}$ indicate Bottle pickup and target-tray placement in the same rollout. Phone progress credits partial completion; Bottle success requires both stages. Appendix C.3 provides the per-rollout Phone stage outcomes and Phone full-task success.

C Detailed Main Results

This appendix reports RoboTwin task-level results, Florence–SimplerEnv training-seed robustness, and physical Phone rollout outcomes. Continuation and evaluation protocols are specified in Appendix B.

C.1 RoboTwin Task-Level Results

The RoboTwin tables retain all 20 tasks in the fixed evaluation subset. Each entry reports Easy / Hard successes, with 25 rollouts for each task and condition. The total row separately sums the 500 Easy and 500 Hard outcomes. Tasks with zero successes remain visible so that the evaluated panel and its aggregation are fully specified.

Main-comparison methods. Tables 7 and 8 report the task-level outcomes for Uniform, FrameSkip, KF-Weight, Loss-IS, and ReTime on Florence–RoboTwin and Qwen–RoboTwin, respectively.

Table 7: Task-level Florence–RoboTwin strict-success counts (Easy/Hard; 25 rollouts per task and condition). Totals are out of 500 per condition.

Task	Uniform	FrameSkip	KF-Weight	Loss-IS	ReTime
Adjust bottle	8 / 11	14 / 16	9 / 10	11 / 9	19 / 18
Click bell	23 / 21	25 / 20	14 / 18	25 / 23	19 / 15
Open laptop	13 / 14	13 / 10	4 / 7	11 / 11	20 / 15
Turn switch	13 / 9	14 / 9	12 / 9	6 / 11	12 / 10
Beat block with hammer	1 / 0	5 / 3	2 / 5	1 / 8	3 / 4
Rank blocks (RGB)	0 / 1	0 / 0	0 / 1	0 / 0	0 / 0
Handover block	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
Handover microphone	0 / 1	0 / 0	3 / 5	0 / 0	1 / 3
Pick dual bottles	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
Dump bin into big bin	7 / 9	13 / 11	5 / 5	14 / 12	11 / 10
Grab roller	3 / 12	21 / 16	18 / 18	6 / 8	24 / 22
Hang mug	0 / 1	0 / 2	0 / 0	0 / 0	0 / 0
Lift pot	0 / 0	5 / 5	1 / 0	0 / 1	4 / 3
Shake bottle	17 / 16	23 / 20	17 / 15	18 / 18	23 / 22
Place A to B (left)	7 / 9	7 / 11	4 / 4	7 / 10	9 / 10
Place bread in basket	2 / 2	4 / 3	3 / 1	3 / 0	6 / 5
Place cans in plastic box	0 / 0	2 / 4	0 / 0	1 / 0	0 / 1
Put object in cabinet	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
Rotate QR code	3 / 4	5 / 5	2 / 3	4 / 0	1 / 0
Stack three blocks	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
Total	**97 / 110**	**151 / 135**	**94 / 101**	**107 / 111**	**152 / 138**

Table 8: Task-level Qwen–RoboTwin strict-success counts (Easy/Hard; 25 rollouts per task and condition). Totals are out of 500 per condition.

Task	Uniform	FrameSkip	KF-Weight	Loss-IS	ReTime
Adjust bottle	0 / 1	0 / 0	0 / 0	0 / 0	1 / 0
Click bell	12 / 12	15 / 10	13 / 13	16 / 13	14 / 14
Open laptop	5 / 8	3 / 5	3 / 3	6 / 6	7 / 6
Turn switch	8 / 7	7 / 6	6 / 5	8 / 7	9 / 8
Beat block with hammer	0 / 0	0 / 0	0 / 0	1 / 0	0 / 0
Rank blocks (RGB)	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
Handover block	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
Handover microphone	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
Pick dual bottles	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
Dump bin into big bin	0 / 0	0 / 0	0 / 1	1 / 0	0 / 3
Grab roller	7 / 5	9 / 8	4 / 3	8 / 7	10 / 5
Hang mug	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
Lift pot	0 / 0	0 / 0	0 / 0	0 / 0	2 / 2
Shake bottle	8 / 6	10 / 7	10 / 4	5 / 9	11 / 8
Place A to B (left)	0 / 1	1 / 0	0 / 0	0 / 0	1 / 0
Place bread in basket	0 / 0	0 / 1	0 / 0	0 / 0	1 / 0
Place cans in plastic box	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
Put object in cabinet	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
Rotate QR code	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
Stack three blocks	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
Total	**40 / 40**	**45 / 37**	**36 / 29**	**45 / 42**	**56 / 46**

C.2 Florence–SimplerEnv Training-Seed Robustness

To assess whether the primary Florence–SimplerEnv result depends on a particular training seed, we repeat Uniform, Shuffle, and ReTime under three training seeds while keeping the continuation configuration and evaluation protocol fixed. All checkpoints are evaluated using the same four evaluation seeds, with 96 rollouts per task and 384 rollouts overall. Table 9 reports the complete results.

Table 9: Florence–SimplerEnv strict success across three training seeds after 500 updates. Task rates use 96 rollouts and Overall uses 384 (%, ↑); all checkpoints share the same evaluation seeds.

Training seed	Method	Stack	Carrot	Spoon	Eggplant	Overall
117465	Uniform	13.54	32.29	52.08	70.83	42.19
	Shuffle	7.29	52.08	68.75	90.63	54.69
	ReTime	25.00	46.88	86.46	79.17	59.38
533389	Uniform	16.67	28.13	29.17	97.92	42.97
	Shuffle	29.17	56.25	48.96	83.33	54.43
	ReTime	37.50	60.42	72.92	82.29	63.28
944964	Uniform	28.13	54.17	59.38	92.71	58.59
	Shuffle	18.75	50.00	77.08	90.63	59.11
	ReTime	34.38	40.63	80.21	90.63	61.46
Overall mean ± s.d.						
	Uniform					47.92 ± 9.25
	Shuffle					56.08 ± 2.63
	ReTime					$\mathbf{61.37 \pm 1.95}$

The same aggregate ordering is observed for all three training seeds: ReTime outperforms Shuffle, which in turn outperforms Uniform. Across the three seeds, the overall strict-success rates are

$61.37 \pm 1.95\%$ for ReTime, $56.08 \pm 2.63\%$ for Shuffle, and $47.92 \pm 9.25\%$ for Uniform (mean $\pm$ sample standard deviation). Computed from the integer success counts before rounding, the mean improvements over Uniform and Shuffle are 13.45 and 5.30 percentage points, respectively. These results support robustness to the three tested training seeds within Florence–SimplerEnv; they do not establish the same variability for other settings or baselines. The main table reports seed 117465, whose outcomes are 42.19%, 54.69%, and 59.38% for Uniform, Shuffle, and ReTime, respectively.

C.3 Real-World Results

Tables 10 and 11 report the ten Phone rollouts per method with the Florence and Qwen backbones, respectively. Each entry contains the three binary stage outcomes and their sum. The hardware platform, task sequence, and metric definitions are given in Appendix B.7.

With Florence, ReTime obtains 25/30 points (83.3%), compared with 21/30 (70.0%) for Loss-IS; with Qwen, it obtains 27/30 (90.0%), compared with 22/30 (73.3%) for FrameSkip and Loss-IS. The normalized-score gains are 13.3 and 16.7 percentage points.

Full-task success is summarized for every method in the raw-result tables. With Florence, ReTime completes 6/10 rollouts, compared with 3/10 for Loss-IS, the strongest baseline by both stage score and full-task success. With Qwen, ReTime completes 7/10 rollouts, compared with 4/10 for FrameSkip and 5/10 for Loss-IS, the two strongest baselines by stage score. These outcomes describe the fixed Phone panels. Bottle is reported through aggregate binary full-task success in Table 1.

Table 10: Per-rollout Phone results with Florence. The tuple records pickup, placement, and lid-closing success; the bold value is the resulting 0–3 stage score.

Rollout	Starting Policy	Uniform	FrameSkip	KF-Weight	Loss-IS	ReTime
1	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,1)$ **3**
2	$(1,1,0)$ **2**	$(1,1,0)$ **2**	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,1,1)$ **3**	$(1,1,1)$ **3**
3	$(1,0,0)$ **1**	$(1,1,1)$ **3**	$(1,1,1)$ **3**	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,1,0)$ **2**
4	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,1)$ **3**
5	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,1)$ **3**	$(1,1,0)$ **2**
6	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,0)$ **2**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,1)$ **3**
7	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,1)$ **3**	$(1,1,1)$ **3**	$(1,1,0)$ **2**	$(1,1,0)$ **2**
8	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,1)$ **3**
9	$(0,0,0)$ **0**	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,0,0)$ **1**
10	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,1)$ **3**
Total	**13/30 (43.3%)**	**18/30 (60.0%)**	**20/30 (66.7%)**	**17/30 (56.7%)**	**21/30 (70.0%)**	**25/30 (83.3%)**
Full task	0/10 (0%)	2/10 (20%)	2/10 (20%)	2/10 (20%)	3/10 (30%)	6/10 (60%)

Table 11: Per-rollout Phone results with Qwen. The tuple records pickup, placement, and lid-closing success; the bold value is the resulting 0–3 stage score.

Rollout	Starting Policy	Uniform	FrameSkip	KF-Weight	Loss-IS	ReTime
1	$(1,0,0)$ **1**	$(1,1,1)$ **3**	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,1,1)$ **3**	$(1,1,1)$ **3**
2	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,1,1)$ **3**
3	$(1,1,0)$ **2**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,0,0)$ **1**	$(1,1,1)$ **3**	$(1,1,1)$ **3**
4	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,0)$ **2**
5	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,1)$ **3**
6	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,0)$ **2**	$(1,1,1)$ **3**
7	$(1,1,1)$ **3**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,0)$ **2**
8	$(0,0,0)$ **0**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,1,0)$ **2**
9	$(1,1,0)$ **2**	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,1,1)$ **3**	$(1,1,1)$ **3**
10	$(0,0,0)$ **0**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,1,1)$ **3**
Total	**14/30 (46.7%)**	**17/30 (56.7%)**	**22/30 (73.3%)**	**14/30 (46.7%)**	**22/30 (73.3%)**	**27/30 (90.0%)**
Full task	1/10 (10%)	1/10 (10%)	4/10 (40%)	1/10 (10%)	5/10 (50%)	7/10 (70%)

Table 12: Per-rollout Florence–Phone results from the shared epoch-20 state. The tuple records pickup, placement, and lid-closing success; the bold value is the resulting 0–3 stage score. Uniform continues on the parent trajectory; ReTime resumes the saved state. Epoch 40 is the main comparison in Table 1.

	Start	**Uniform**				**ReTime**			
Rollout	**20**	**30**	**40**	**50**	**60**	**30**	**40**	**50**	**60**
1	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,0)$ **2**	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,1)$ **3**	$(1,1,1)$ **3**
2	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,1)$ **3**	$(1,1,1)$ **3**	$(1,1,0)$ **2**
3	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,1,1)$ **3**	$(1,1,1)$ **3**	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,1)$ **3**
4	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,1,1)$ **3**	$(1,1,1)$ **3**	$(1,1,1)$ **3**	$(1,1,1)$ **3**	$(1,1,1)$ **3**
5	$(1,0,0)$ **1**	$(0,0,0)$ **0**	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,1,1)$ **3**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,1)$ **3**
6	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,0)$ **2**	$(1,1,1)$ **3**
7	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,1)$ **3**	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,1,1)$ **3**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,0)$ **2**
8	$(1,1,0)$ **2**	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,0)$ **2**	$(1,1,0)$ **2**	$(1,1,1)$ **3**	$(1,1,0)$ **2**	$(1,1,0)$ **2**
9	$(0,0,0)$ **0**	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,1,1)$ **3**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,1,0)$ **2**
10	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,1,0)$ **2**	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,0,0)$ **1**	$(1,1,1)$ **3**	$(1,0,0)$ **1**	$(1,1,1)$ **3**
Total	**13/30**	**15/30**	**18/30**	**19/30**	**17/30**	**21/30**	**25/30**	**25/30**	**26/30**
%	**43.3**	**50.0**	**60.0**	**63.3**	**56.7**	**70.0**	**83.3**	**83.3**	**86.7**

C.3.1 Florence–Phone Training-Horizon Sweep

Table 12 reports matched continuation horizons from a common epoch-20 state. Uniform is trained continuously from epoch 0, so its later states require no explicit restoration. ReTime resumes from the same epoch-20 state after 17,460 optimizer updates and restores the model parameters, optimizer state, learning-rate scheduler, and random-number-generator states. Each method receives 10, 20, 30, or 40 additional epochs and is evaluated with ten rollouts, for a maximum accumulated score of 30 points. ReTime scores 21, 25, 25, and 26 points, whereas Uniform scores 15, 18, 19, and 17 points; the corresponding normalized-score gaps are 20.0, 23.3, 20.0, and 30.0 percentage points. Scores measure stage progress rather than binary full-task success, and the sweep is limited to Florence–Phone.

D Qualitative Execution Examples

Figure 10 shows selected FrameSkip and ReTime rollouts in SimplerEnv stacking and real-world Phone manipulation. In the stacking example, both methods complete grasp and transport, while the FrameSkip rollout fails at placement after target misalignment. In the Phone example, FrameSkip fails to establish a stable grasp after approaching the object, while ReTime proceeds through the shown manipulation stages. The examples illustrate execution differences; they do not estimate failure frequencies or explain the aggregate success differences on their own.

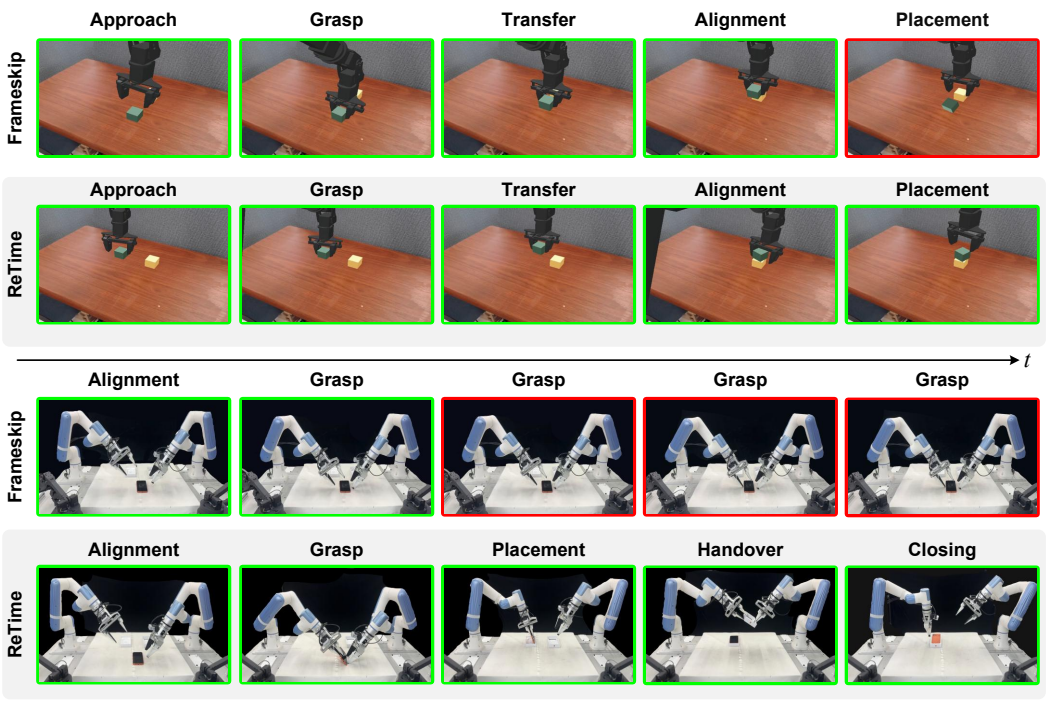


Figure 10: Illustrative FrameSkip (upper row) and ReTime (lower row) rollouts. Top: in SimplerEnv stacking, FrameSkip grasps and transports the object but fails placement after target misalignment; ReTime completes the shown stages. Bottom: in real-world Phone manipulation, FrameSkip does not establish a stable grasp, while ReTime proceeds through closing. Green and red borders mark successful and failed stage observations. These selected cases are not estimates of failure-mode frequency.

E Factorial Mechanism Analysis

This appendix reports the complete numerical results underlying Figure 4. The analysis crosses two factors. The temporal-assignment factor either preserves the correspondence between residual scores and physical-time anchors (Residual-aligned) or randomly permutes the same calibrated proposal masses within each trajectory (Shuffled assignment). The objective-treatment factor either uses sampled losses directly (No IC) or applies the ordinary p/q importance correction (p/q IC). With importance correction enabled, the Shuffled and Residual-aligned assignments correspond to Shuffle and ReTime, respectively. Uniform is included as a reference; because $q = p$ in that condition, its importance weight is identically one.

All four factorial cells use $K = 8$, $\alpha = 0.2$, and 500 continuation updates. Within each backbone–benchmark setting, they otherwise follow the shared continuation and evaluation protocol in Appendix B.

E.1 Complete Factorial Results

Table 13 reports all four cells together with the Uniform reference. Comparisons between the two temporal assignments hold objective treatment fixed, while comparisons within an assignment hold the temporal proposal fixed and change only importance correction.

Interpretation of the controls. Within the importance-corrected branch, the Uniform–Shuffle contrast tests the non-uniform temporal proposal after residual–time correspondence is removed, and the Shuffle–ReTime contrast tests residual assignment while retaining the per-trajectory multiset of temporal-proposal probabilities and importance weights. The aggregate results support both non-uniform visitation and an additional role for assigning proposal masses to residual-indicated mo-

Table 13: Factorial mechanism results after 500 updates: strict success (%, ↑) by temporal assignment and importance correction (IC). SimplerEnv entries use 96 rollouts per task; RoboTwin entries use 500 per condition. Bold marks the best result in each row.

Setting	Metric	Start	Uniform	Shuffle		Residual-aligned	
		$q{=}p$	$q{=}p$	No IC	p/q	No IC	p/q
Florence–SimplerEnv	Stack	4.17	13.54	18.75	7.29	**30.21**	25.00
	Carrot	12.50	32.29	51.04	**52.08**	46.88	46.88
	Spoon	44.79	52.08	61.46	68.75	69.79	**86.46**
	Eggplant	84.38	70.83	80.21	**90.63**	71.88	79.17
	Overall	36.46	42.19	52.86	54.69	54.69	**59.38**
Florence–RoboTwin	Easy	27.40	19.40	25.60	29.20	29.20	**30.40**
	Hard	28.40	22.00	22.00	27.40	27.00	**27.60**
	Overall	27.90	20.70	23.80	28.30	28.10	**29.00**
Qwen–SimplerEnv	Stack	**10.42**	**10.42**	6.25	6.25	**10.42**	9.38
	Carrot	30.21	27.08	29.17	31.25	30.21	**35.42**
	Spoon	23.96	29.17	31.25	**37.50**	**37.50**	30.21
	Eggplant	70.83	85.42	83.33	79.17	70.83	**89.58**
	Overall	33.85	38.02	37.50	38.54	37.24	**41.15**
Qwen–RoboTwin	Easy	7.60	8.00	8.80	9.40	8.80	**11.20**
	Hard	5.80	8.00	8.80	8.60	6.80	**9.20**
	Overall	6.70	8.00	8.80	9.00	7.80	**10.20**

ments. These two contrasts are not an additive causal decomposition of the total Uniform–ReTime difference, and the interpretation does not require every individual task to improve.

Holding temporal assignment fixed compares corrected and uncorrected continuation training. Uncorrected sampling targets the proposal-induced q-weighted objective. Ordinary p/q importance correction restores the baseline objective in expectation while retaining the altered encounter frequencies. Importance correction improves aggregate success for both temporal assignments in all four evaluated settings. Because it changes both objective weights and stochastic-gradient sampling, the performance difference does not isolate the effect of objective drift. It does show that the observed gains are not explained solely by replacing the baseline objective with a residual-weighted one. Objective preservation concerns the baseline continuation objective and its expected raw gradient at fixed parameters; it does not imply identical finite-budget optimization paths.

E.2 Paired Florence–SimplerEnv Outcomes

Evaluation outcomes are paired by continuation-training seed, evaluation seed, task, and rollout ID. The three training seeds reuse the same 384 evaluation scenarios. Table 14 retains the training-seed groups rather than treating the repeated scenarios as independent observations.

Table 14: Paired Florence–SimplerEnv outcomes grouped by training seed (384 per row). R = ReTime, S = Shuffle; Δ is R minus S in percentage points, computed from integer counts.

Training seed	Both succeed	R only	S only	Both fail	Δ
117465	167	61	43	113	4.69
533389	157	86	52	89	8.85
944964	180	56	47	101	2.34

The paired contrasts favor Residual-aligned assignment in each of the three training seeds. These are repeated evaluations on a fixed scenario panel, so the seed groups, rather than the pooled rollout count, describe the available replication.

E.3 Shuffle Permutation Sensitivity

The Shuffle control preserves the complete multiset of calibrated proposal masses within each trajectory and changes only their assignment to physical-time anchors. We repeat the Florence–SimplerEnv Shuffle condition with three independently generated temporal permutations. All runs use the shared state after 20,000 optimizer updates, $K = 8$, $\alpha = 0.2$, the 500-update continuation budget, and the evaluation panel specified in Appendix B.

Table 15: Florence–SimplerEnv sensitivity of Shuffle to three temporal permutations at fixed training seed 117465. Task entries are successes out of 96; Overall is out of 384. Assignment 1 is used in the factorial study.

Permutation	Stack	Carrot	Spoon	Eggplant	Overall
149789841	7	50	66	87	210/384 (54.69%)
1493795941	16	58	56	75	205/384 (53.39%)
176876391	27	47	59	71	204/384 (53.13%)
Mean $\pm$ s.d.					53.73% $\pm$ 0.84

The three permutations obtain 54.69%, 53.39%, and 53.13% strict success (mean 53.73%; sample standard deviation 0.84 percentage points). This limited sensitivity check concerns Florence–SimplerEnv only.

F Residual-Signal and Proposal Diagnostics

This appendix defines the diagnostics summarized in Section 4.3. The analysis uses fixed 512-trajectory diagnostic panels for Florence and Qwen and $K = 8$ temporal anchors per trajectory, yielding 4,096 audited anchors for each backbone at its respective shared starting checkpoint. Residual reliability is measured from independent estimates of the frozen starting policy, while optimization relevance is evaluated against a separately estimated action-model gradient target. These panels diagnose the sampled sparse-anchor support and are not estimates over every valid training anchor.

F.1 Residual-Estimation Reliability

The residual estimator in Appendix A.1 produces four independent estimates for every anchor. We split these estimates into two nonoverlapping pairs and average within each pair. The two resulting score vectors are converted independently to within-trajectory ranks before their agreement is measured. We report the Spearman correlation after pooling the within-trajectory ranks, the median correlation computed separately over the 512 trajectories, and the mean overlap between the two highest-ranked anchors from each estimate.

This construction measures whether repeated residual estimation recovers the same temporal ordering. Using within-trajectory ranks removes differences in absolute residual scale across trajectories and matches the calibration used to construct the ReTime proposal.

F.2 Independent Gradient-Utility Target

For each audited anchor i, we estimate an action-model gradient target using random streams disjoint from those used for residual scoring. For estimator replicate r and Flow-Matching draw j, let

$$\ell_{irj} = \mathrm{mean}\Big[\big(v_\theta(x_{irj}, \tau_{irj}, c_i) - u_{irj}\big)^2\Big], \qquad g_{irj} = \|\nabla_{\theta_{\mathrm{act}}} \ell_{irj}\|_2 , \tag{26}$$

where θ_{act} contains the action-model parameters. The VLM conditioning is computed without gradients and detached before the action model is evaluated. The target therefore measures action-model gradient magnitude rather than the gradient of the full vision-language backbone.

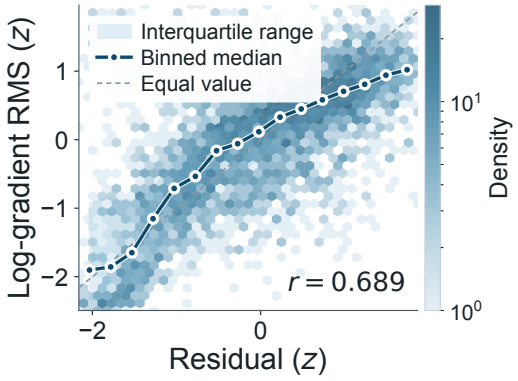


Figure 11: **Residual–gradient alignment on the Florence diagnostic panel.** The audit contains 4,096 temporal anchors from 512 trajectories ($K = 8$ per trajectory); 3,949 remain after clipping the outer 1% of each plotted axis for visualization. Residual scores and log action-model gradient RMS are standardized within each trajectory before pooling. Hexagon intensity indicates anchor density, while the blue curve and band show binned medians and interquartile ranges. Higher residual scores are associated with larger independently estimated action-model gradient magnitudes ($r = 0.689$ after clipping).

We aggregate the gradient norms with two root-mean-square reductions:

$$e_{ir} = \sqrt{\frac{1}{J}\sum_{j=1}^{J} g_{irj}^2}, \qquad h_i = \sqrt{\frac{1}{R}\sum_{r=1}^{R} e_{ir}^2}, \tag{27}$$

using $J = 8$ Beta-stratified, antithetically paired time–noise draws and $R = 4$ independent estimator replicates. The resulting h_i is used only as an audit target and does not enter proposal construction or continuation training. Because it averages the squared norms of per-draw gradients, it differs from the squared norm of a gradient averaged over multiple draws, which would additionally contain cross terms. The target should therefore be interpreted under the per-draw definition in Equation 27.

F.3 Within-Trajectory Alignment

Residual scores and gradient targets are converted independently to within-trajectory percentile ranks. The pooled residual–gradient Spearman correlation is computed after concatenating these ranks across all audited anchors. We also compute a Spearman correlation separately within each trajectory and report its median and the number of trajectories with strictly positive correlation.

For top-2 overlap, let $\mathcal{T}^{(s)}_{g,\mathrm{top2}}$ and $\mathcal{T}^{(h)}_{g,\mathrm{top2}}$ denote the two highest-ranked residual and gradient-target anchors in trajectory g. We report

$$O_g = \frac{\left|\mathcal{T}^{(s)}_{g,\mathrm{top2}} \cap \mathcal{T}^{(h)}_{g,\mathrm{top2}}\right|}{2} \tag{28}$$

averaged equally over trajectories. Figure 11 provides a complementary visualization of the residual–gradient relationship on the Florence diagnostic panel. After removing trajectory-specific scale differences, the pooled anchors exhibit a clear increasing trend: moments with larger-than-average residuals within a trajectory also tend to produce larger-than-average action-model gradient magnitudes.

F.4 Gradient Second-Moment Proxy

For an audited trajectory g with n_g audited anchors, let $p_{gi} = 1/n_g$ be the probability assigned by the baseline temporal distribution, q_{gi} the temporal-proposal probability, and h_{gi} the gradient target

Table 16: Residual-signal diagnostics at the shared starting checkpoints (512 trajectories and 4,096 anchors per backbone). Higher is better except for the second-moment ratio.

Statistic	Florence	Qwen
Residual reliability		
Pooled rank ρ	0.9621	0.9596
Median trajectory rank ρ	0.9762	0.9762
Top-2 overlap	0.8965	0.9307
Optimization relevance		
Pooled residual–gradient ρ	0.6529	0.7615
Median trajectory residual–gradient ρ	0.8095	0.8333
Residual–gradient top-2 overlap	0.5811	0.7500
Positive-correlation trajectories	463/512 (90.43%)	498/512 (97.27%)
Improved second-moment trajectories	430/512 (83.98%)	485/512 (94.73%)
Second-moment ratio vs. Uniform	0.97114	0.92428

from Equation 27. The plug-in second moment of the importance-corrected per-anchor gradient is

$$\widehat{M}_g(q_g) = \sum_{i=1}^{n_g} \frac{p_{gi}^2 h_{gi}^2}{q_{gi}}, \qquad \widehat{M}_g(p_g) = \frac{1}{n_g} \sum_{i=1}^{n_g} h_{gi}^2. \tag{29}$$

The factor p_{gi}^2/q_{gi} follows from sampling under q_{gi} and multiplying each gradient by p_{gi}/q_{gi}. We aggregate trajectories with equal weight:

$$R_M = \frac{\frac{1}{G}\sum_{g=1}^{G} \widehat{M}_g(q_g)}{\frac{1}{G}\sum_{g=1}^{G} \widehat{M}_g(p_g)}. \tag{30}$$

A trajectory is counted as improved when $\widehat{M}_g(q_g)/\widehat{M}_g(p_g) < 1$. Table 16 reports the complete reliability, alignment, and second-moment statistics.

Interpretation of reliability and gradient alignment. Agreement between independent residual estimates tests whether temporal rankings repeat despite estimator randomness. Rank correlations near 0.96 and Top-2 overlaps above 0.89 indicate that the ranking differences are repeatable under the evaluated estimation protocol. Residual–gradient correlations of 0.653 for Florence and 0.762 for Qwen show a positive association with the independently estimated action-model gradient-magnitude target. Larger residuals therefore tend to identify anchors with larger values of that target, although residual magnitude alone does not determine a full-policy gradient.

On the audited sparse-anchor support, the residual-guided temporal proposal lowers the gradient second-moment proxy on 84.0% of Florence trajectories and 94.7% of Qwen trajectories. Qwen also has stronger residual–gradient alignment and a lower aggregate relative second moment (0.924, compared with 0.971 for Florence). The joint pattern is consistent with a relationship between temporal ranking quality and the sampling proxy in these two settings. It does not establish a causal relationship across backbones or guarantee lower full-policy gradient variance throughout continuation training. Downstream task performance is assessed by the separate continuation experiments.

A fixed-checkpoint comparison on the same 512 Florence trajectories at +500 updates finds that freshly rescoring the proposal lowers the aggregate second-moment proxy by 0.18% relative to the frozen proposal. This diagnostic does not evaluate a continuation arm with periodic rescoring.

G Proposal Ablations

This appendix reports the detailed ablations of the two proposal parameters studied in Section 4.4. The mixture coefficient α controls how far the temporal proposal departs from the baseline temporal distribution, whereas K controls the temporal resolution and cost of residual estimation. Both studies use Florence–SimplerEnv continuation training from the shared state after 20,000 optimizer updates.

Table 18: Sparse-to-Dense fidelity on 512 Florence–SimplerEnv trajectories (19,375 anchors). Rank ρ and top-10% recall compare residual orderings. $M(q)/M(p)$ is the second-moment ratio versus Uniform; recovered is the share of the Dense reduction.

K	Rank ρ	Top 10%	Anchors	Cost	$M(q)/M(p)$	Recovered
4	0.8354	0.4029	2,048	10.6%	0.978959	79.37%
8	0.9359	0.5954	4,096	21.1%	0.974865	94.81%
16	0.9748	0.7806	8,177	42.2%	0.973859	98.61%
Dense	1.0000	1.0000	19,375	100%	0.973490	100%

G.1 Residual-Guidance Strength

We fix $K = 8$ and $\delta = 0.1$ and vary $\alpha \in \{0.1, 0.2, 0.3, 0.4\}$. Every setting uses ordinary p/q importance correction and 500 exact-state continuation updates. Evaluation covers Stack, Carrot, Spoon, and Eggplant, with 96 rollouts per task and 384 rollouts in total.

The support floor in Equation 18 gives the importance-weight bound

$$\frac{p(t \mid e, d)}{q_\alpha(t \mid e, d)} \leq \frac{1}{1-\alpha}. \tag{31}$$

Table 17 reports this bound together with the task-level outcomes.

Table 17: Florence–SimplerEnv sensitivity to the mixture coefficient α at $K = 8$ and training seed 117465. Task entries are successes out of 96; Overall is out of 384. Bold entries mark column maxima; the bold α marks the default.

α	Weight bound	Stack	Carrot	Spoon	Eggplant	Overall
0.1	1.1111	33	53	68	75	**229**
0.2	1.2500	24	45	**83**	**76**	228
0.3	1.4286	**42**	50	75	59	226
0.4	1.6667	25	**55**	73	67	220

The two moderate settings, $\alpha = 0.1$ and $\alpha = 0.2$, differ by one success in aggregate, while $\alpha = 0.4$ is 2.34 percentage points below $\alpha = 0.1$. Task-level rankings differ; we use $\alpha = 0.2$ as the fixed default for the reported experiments.

G.2 Temporal Scoring Resolution

The resolution study evaluates $K \in \{4, 8, 16\}$ against Dense residual scoring on a fixed panel of 512 trajectories. Dense scoring evaluates all 19,375 valid anchors. Each sparse proposal retains the corresponding evenly spaced anchor scores from the Dense cache, linearly interpolates the remaining valid positions, and applies the same rank calibration, $\delta = 0.1$, and $\alpha = 0.2$ as the main method.

G.2.1 Proposal and Optimization Fidelity

Table 18 compares sparse scoring with Dense on the same 512-trajectory panel. Rank correlation and top-10% recall measure agreement with the Dense residual ordering; scored anchors count policy evaluations before interpolation. The second-moment ratio uses the independent per-draw action-model gradient target in Appendix F.2 over all 19,375 valid anchors, rather than the $K = 8$ audited support of Appendix F.4. The recovered reduction is $100(1 - R_K)/(1 - R_D)$, where $R_K = M(q^K)/M(p)$ and $R_D = M(q^{\mathrm{Dense}})/M(p)$ follow Equation 30. These are offline proxies, not measurements of full-policy gradient variance or downstream success.

Increasing K improves offline agreement and the second-moment proxy while increasing residual-scoring work. At $K = 8$, the proposal uses 21.1% of Dense evaluations and recovers 94.81% of its proxy reduction.

Table 19: Florence–SimplerEnv strict success after 500 updates at training seed 117465. Each task uses 128 rollouts; Overall uses 512. Entries are successes, with the rate in parentheses. Bold marks the best result in each row.

Task	$K = 4$	$K = 8$	$K = 16$
Stack	**49** (38.28%)	38 (29.69%)	24 (18.75%)
Carrot	**85** (66.41%)	70 (54.69%)	71 (55.47%)
Spoon	97 (75.78%)	**111** (86.72%)	91 (71.09%)
Eggplant	81 (63.28%)	**99** (77.34%)	92 (71.88%)
Overall	312 (60.94%)	**318** (62.11%)	278 (54.30%)

G.2.2 Downstream Continuation

The downstream comparison changes only K in the Florence–SimplerEnv +500 exact-state continuation training. Each setting is evaluated with 128 rollouts per task, yielding 512 rollouts in total.

The 512-rollout panel ranks $K = 8 > K = 4 > K = 16$ in aggregate, although task-level rankings differ. Dense scoring has no downstream continuation result. Better offline fidelity at $K = 16$ therefore does not imply better task success in this comparison.

H Performance–Compute Trade-off on Florence–Phone

This appendix specifies the additional-compute accounting summarized in Section 4.5. The comparison uses Florence–Phone. All methods use the same epoch-20 starting state and receive another 20 epochs (17,460 optimizer updates) on eight GPUs. Uniform training continues uninterrupted from this state, while the other methods restore it before continuation; this operational difference does not change the common comparison state or subsequent update budget. The shared continuation workload costs 29.63 wall-clock hours, so Table 21 reports only computation beyond that common cost. Phone task progress is taken from the real-world evaluation in Table 1; the physical-robot experiments are not rerun for this analysis. Separate simulation timing estimates are reported in Appendix I.3.

Online timing protocol. The continuation window runs from optimizer update 17,460 to update 34,920, for 17,460 updates with global batch size 128 on eight GPUs. Starting from the state after 17,460 optimizer updates, we time 10 updates for each method. Each update is bracketed by CUDA synchronizations, includes batch retrieval, and records the slowest wall-clock time among the eight ranks. We discard the first three updates as warmup and use the median of the remaining seven. Table 20 reports these retained measurements; in particular, the Loss-IS estimate is based on seven measured updates (10 total, three discarded), rather than treating the 10-update probe itself as the reported cost.

Table 20: Retained per-update wall-clock measurements for the Florence–Phone probe. Values are seconds on eight GPUs after discarding three warmup updates; individual measurements are rounded to four decimal places and medians to five.

Method	**Seven retained measurements (s)**	**Median (s)**
Uniform	6.1088, 6.0716, 6.0943, 6.5483, 6.4887, 6.0992, 6.1482	6.10882
ReTime	6.1979, 6.1811, 6.1943, 6.9099, 6.5220, 6.1021, 6.0915	6.19430
FrameSkip	6.0445, 6.1080, 6.1147, 6.8414, 6.5982, 6.1369, 6.2186	6.13695
Loss-IS	52.0014, 47.9710, 48.8946, 46.6378, 49.0676, 49.3417, 48.5842	48.89459

The Uniform median defines the shared continuation time and the zero of method-specific additional cost:

$$T_{\text{shared}} \approx 17{,}460 \times 6.10882/3600 \approx 29.6278 \text{ wall-h}. \tag{32}$$

Loss-IS uses online scoring with a candidate pool three times the backward-batch size. Its median includes both training and candidate scoring. We therefore subtract the Uniform median and extrapolate only the per-update difference over the continuation horizon:

$$\Delta t_{\text{Loss-IS}} \approx 48.89459 - 6.10882 = 42.78577 \text{ s/update}, \tag{33}$$

$$T_{\text{extra}} = 17{,}460\, \Delta t_{\text{Loss-IS}}/3600 \approx 207.511 \text{ h}, \tag{34}$$

$$G_{\text{extra}} = 8T_{\text{extra}} \approx 1660.088 \text{ GPU-h}. \tag{35}$$

Thus, the reported 1660.088 GPU-hours are an extrapolation from the seven retained probe updates, not the cost of executing the 10-update probe. The factor of eight converts the measured eight-GPU wall-clock overhead to aggregate device time. Individual readings and offline means are displayed to four decimal places; the two online medians in the extrapolation are displayed to five because the per-update difference is multiplied over 17,460 updates. All reported totals are calculated from the unrounded measurements.

Offline timing protocol. Offline preprocessing is timed on individual episodes or trajectories. For ReTime and FrameSkip, we measure 16 items, discard the first to exclude CUDA startup, and average the remaining 15 before projecting to all 305 Phone demonstrations. These one-time costs are not multiplied by the continuation-update count. They are measured as GPU-hours; eight-way parallelism divides their projected wall-clock time by eight without reducing aggregate GPU-hours.

ReTime uses $K = 8$, four estimator seeds, and eight samples per scored anchor. Its retained 15-episode end-to-end mean, including image loading, is 11.3757 seconds per episode. Its projected costs are

$$G_{\text{ReTime}} \approx 305 \times 11.3757/3600 = 0.9638 \text{ GPU-h}, \tag{36}$$

$$T_{\text{ReTime}} = G_{\text{ReTime}}/8 = 0.1205 \text{ wall-h}. \tag{37}$$

FrameSkip times DINOv2-S/14 feature extraction together with video decoding. Its retained 15-trajectory mean is 7.3128 seconds per trajectory, giving

$$G_{\text{FrameSkip}} \approx 305 \times 7.3128/3600 = 0.6196 \text{ GPU-h}, \tag{38}$$

$$T_{\text{FrameSkip}} = G_{\text{FrameSkip}}/8 = 0.0774 \text{ wall-h}. \tag{39}$$

For KF-Weight, one measured predictor-training epoch takes 1.5206 seconds, and one full-support action-prediction-error pass takes 5.6205 seconds. Extrapolating the predictor fit to 300 epochs gives

$$G_{\text{KF}} \approx (300 \times 1.5206 + 5.6205)/3600 = 0.1283 \text{ GPU-h}, \tag{40}$$

$$T_{\text{KF}} = G_{\text{KF}}/8 = 0.0160 \text{ wall-h}. \tag{41}$$

Uniform has no method-specific preprocessing, so its additional cost is defined as zero.

Table 21: Additional adaptation cost on Florence–Phone. All methods share the same 17,460-update continuation workload (29.63 wall-clock hours on eight GPUs). Phone reports normalized task progress from Table 1; additional compute is projected from timing probes and excludes the shared continuation. Wall-clock overhead assumes eight-way parallel execution.

Method	Phone (%)	GPU-h	Wall-clock (h)
Uniform	60.00	0.000	0.000
FrameSkip	66.67	0.620	0.077
KF-Weight	56.67	0.128	0.016
Loss-IS	70.00	1660.088	207.511
ReTime	**83.33**	0.964	0.120

Table 21 reports the measured-task outcomes alongside projected method-specific overhead. ReTime's cache is built once for the starting-policy–dataset pair, while Loss-IS candidate scoring recurs at every update. Figure 6 plots the eight-GPU wall-clock projections; Uniform has zero method-specific overhead and is omitted from its logarithmic time axis.

I Reproducibility Information

I.1 Reproduction Path

Simulation runs restore the shared starting state in Table 3 and follow the matched continuation and evaluation protocol in Appendix B. Appendix A specifies ReTime's residual scoring, proposal construction, and corrected loss; Appendix B.4 specifies the other methods. Table 6 reports the available optimizer and continuation settings for Florence–Phone; equally detailed records for the other physical settings are not reported here.

I.2 Result Aggregation and Rounding

Success rates sum integer successes and rollouts over the stated evaluation panel before division and rounding; percentage-point differences use unrounded rates. Phone uses normalized stage progress and Bottle uses binary full-task success (Equation 25). Their real-world average weights the two unrounded task metrics equally. Appendix C gives RoboTwin and Phone counts and Florence–SimplerEnv training-seed results; Appendices E and G give aggregate mechanism results and downstream ablations.

I.3 Projected Computational Cost

Table 22 documents ReTime training-update timings and its projected offline cache cost in the four simulation settings. The Florence–Phone comparison uses a separate eight-GPU measurement (Appendix H). Simulation training timings use in-memory synthetic, model-ready inputs with real VLM/DiT computation. After three warm-up updates, we synchronize and time seven updates and report their median. Florence–SimplerEnv, Florence–RoboTwin, Qwen–SimplerEnv, and Qwen–RoboTwin use two, two, one, and two devices, respectively. The Qwen timing probes use AdamW, with DeepSpeed for distributed execution in the RoboTwin probe; these timing configurations differ from the 8-bit AdamW performance configurations in Table 4.

The offline estimate multiplies measured per-anchor scoring time by the full $K = 8$ workload (1,006,912 Bridge or 220,000 RoboTwin anchors) and divides by eight devices. The Qwen unit estimates are approximately 0.070 and 0.074 seconds per anchor for Bridge and RoboTwin. This linear projection does not establish realized eight-device speedup. The timed windows exclude model loading, dataset/video I/O, synthetic-input construction, warm-up/compilation, and cache serialization. These measurements document ReTime's additional computation under the stated timing setup; they are not end-to-end adaptation times or a cross-method compute ranking.

Table 22: ReTime simulation compute measurements. Training reports median seconds per optimizer update with in-memory synthetic inputs on each setting's native device count. Offline cache minutes are linear projections to eight devices and exclude data I/O and serialization.

Setting	Training devices	Train (s/update)	Offline cache (min)
Florence–SimplerEnv	2	4.047	135.2
Florence–RoboTwin	2	7.656	84.3
Qwen–SimplerEnv	1	5.234	146.5
Qwen–RoboTwin	2	1.454	33.9

I.4 Method Implementation Release

Upon publication, we will release the ReTime method implementation as a direct integration with StarVLA, together with instructions for configuring and using it. This artifact implements the method for reuse and is distinct from the internal development version used to conduct the reported experiments. The paper and appendices separately report the available experimental settings, continuation protocols, evaluation panels, integer rollout counts, aggregation rules, and compute-measurement protocols supporting the reported results.